



IBM VisualAge for COBOL for OS/2

GC26-8421-01

Getting Started

IBM

IBM VisualAge for COBOL for OS/2

GC26-8421-01

Getting Started

Note!

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

Second Edition (January 1996)

This edition applies to Corrective Service Disk 1, Version 1 Release 1 of IBM VisualAge for COBOL for OS/2 (part number 28H2177) and to all subsequent releases and modifications until otherwise indicated in new editions. 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 J58
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, 1996. 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	vii
About This Book	ix
How This Book Will Help You	ix
Summary of Changes	xi

Installing VisualAge COBOL	1
Before You Begin	3
Hardware and Software Requirements	3
Installing on LAN-Connected Workstations	5
Installing IBM VisualAge for COBOL for OS/2	7
Changing Your CONFIG.SYS	13
Installing Using Response Files	16
Using a Software Distribution Manager	18
Adding or Deleting Components	19
Deleting Components	20
Installing Additional Components	22
Adding or Deleting Components Using a Response File	23
If Something Goes Wrong	25
If You Get an Error Message	25
If the Component Requires Another Component	25
If You Press Stop	25
If the Install Program Fails	25
If the Second Phase of Installation Fails	26
If the Second Phase of the Installation Does Not Run	27
If You Can't Delete an Object Because It's in Use	27
If VisualAge COBOL Doesn't Appear in the Installation Utility	27
If You Can't Install Additional Components	27
If You've Tried Everything and It Still Doesn't Work	27
Response Files, Options, and Return Codes	29
Response File Format and Parameters	29
Command-Line Options	31
Return Codes	32

Getting Started with VisualAge COBOL	35
Introducing IBM VisualAge for COBOL for OS/2	37

What You Can Do Using VisualAge COBOL	37
Create OS/2 COBOL Applications	37
Create Client/server Applications	37
Create Object-Oriented Applications	38
Your VisualAge COBOL Development Environment	38
Edit, Compile, and Debug	38
Create a GUI for Your Application	39
Create CICS ECI Calls and SQL Statements	39
Access Local and Remote Data	39
Analyze Program Performance	39
Concepts for Developing an Application Using VisualAge COBOL	40
Build Your First VisualAge COBOL Application	43
Creating the Project	43
Creating the Application	45
Building the Application	47
Running the Application	48
Build Your First VisualAge COBOL GUI Application	49
Creating the GUI Project	49
Creating the Graphical User Interface	50
Adding the GUI Parts	52
Customizing the GUI	53
Moving and Sizing GUI Parts	55
Saving the GUI Project	55
Creating the Application Logic	57
Event-Driven Programming	57
Creating the Event Logic	59
Building the Application	65
Running the Application	67
Tools in VisualAge COBOL	69
WorkFrame	69
COBOL Editor	69
COBOL GUI Designer	70
COBOL for OS/2	70
Interactive Debugger	71
Performance Analyzer	72
Tasks and Information for VisualAge COBOL	75
Using the Information with VisualAge COBOL	79
Your Next Step for Learning VisualAge COBOL	79
Using the Online Reference Information	80
Using the Online Help	80
Locating Online Help Topics	80
Printing Online Help Topics	81
Getting VisualAge COBOL Publications	81

Printing Publications	81
Ordering Publications	81
Getting Support for Using VisualAge COBOL	83
Getting Started Period	83
Getting Product Support	83
Getting Consulting Services	84
Getting Education and Training	84

VisualAge COBOL Tutorials 87

Creating a Tax Computation Application with a GUI	89
Creating the Subroutine Project	90
Creating the Subroutine Logic	91
Creating the GUI Application Project	93
Creating the Graphical User Interface (GUI)	94
Adding the GUI Parts	94
Customizing the GUI	94
Saving the GUI Project	95
Creating the Application Logic	97
Calling the Subroutine Logic	100
Nesting the Projects	103
Setting Compiler Options	103
Building the Application	108
Using the Interactive Debugger	110
Preparing Your Application for Debugging	110
Debugging Your Application	110
Ending the Debugging Session	112
Running the Application	112
Packaging the Application for Distribution	113
Installing the Application on an End User Machine	115
Creating SQL Statements with Data Assistant	117
Examining Tables in the Database Schema View	117
Copying Tables to the Data Structure Mapping view	118
Mapping Data Structures	120
Creating an SQL Statement	122

Using the CICS Transaction Assistant 127

Appendix A. Using OS/2 131

Appendix B. VisualAge COBOL Supplied Sample Applications 139

Employee Lookup Application Samples	139
Employee Lookup Application Description	139
Sample Project 1	140
Sample Project 2	141
Sample Project 3	141

Sample Project 4	143
Sample Project 5	144
Sample Project 6	146
SMARTdata Utilities Samples	147
VSAM for the Workstation Samples	147
Data Conversion Utility Samples	148
SMARTsort Samples	148
Appendix C. Configuring APPC Communications	149
Approaching the Task of Configuring Communications	150
Configuring for APPC Communications at CM/2	150
Prerequisites	150
Terminology	150
Communications Manager/2 Configuration Variables	151
Performing CM/2 APPC Configuration	154
Configuring for APPC Communications at MVS	170
Prerequisites	170
Terminology	171
APPC/MVS Configuration Variables	171
APPC/MVS Configuration Overview	174
APPC/MVS Definitions	174
Configuring to Run as a Server	177
Configuring for Remote Debug Tool to Run as a Client	180
Defining the 3745 Attached LAN to VTAM	180
Defining the 3745 Attached LAN to NCP	182
APPC/MVS System Commands	182
Configuring CICS for APPC Communications	183
Prerequisites	184
Terminology	184
CICS Configuration Variables	185
Overview of CICS APPC Configuration	186
Configuring CICS to Run as a Client	186
Defining a Link to the LAN	192
CEMT to Display/Modify Sessions	192
Appendix D. REXX Procedures: Compile and Link	193
REXX Procedures for COBOL Programs	193
REXX Procedures for DB2 Programs	193
Other REXX Procedures	193
VisualAge COBOL Glossary	195
Comparison of Workstation and Mainframe Concepts	199
Index	201

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. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department J01, 555 Bailey Avenue, 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:

ACF/VTAM	DFSMS/MVS
ADSTAR	FFST/2
AIX	First Failure Support Technology/2
APPN	IBM
AS/400	MVS/ESA
CICS	OS/2
CICS/ESA	OS/400
CICS OS/2	Presentation Manager
Common User Access	RS/6000
CUA	SAA
DB2	VisualAge
DB2/2	VSE
DFSMS	VTAM

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

Welcome to IBM VisualAge for COBOL for OS/2, IBM's new COBOL development environment for OS/2! IBM VisualAge for COBOL for OS/2 gives you a comprehensive development environment designed specifically for mission-critical, client/server applications.

IBM VisualAge for COBOL for OS/2 supports local and remote access to DB2, CICS, and VSAM, giving you access to data and transactions nearly anywhere in your enterprise. And all the IBM COBOL family of solutions support the high subset of ANSI 85 COBOL functions, so your applications can be ported across supported platforms, whether they are running on a mainframe, an RS/6000, or a personal computer with OS/2.

IBM VisualAge for COBOL for OS/2 supports object-oriented extensions, allowing you to develop software objects using COBOL and to share SOM-enabled objects created by other languages, like C++.

IBM VisualAge for COBOL for OS/2 provides a complete development environment. The environment includes an editor, debugger, GUI designer, and performance analyzer, all integrated with WorkFrame. WorkFrame integrates your tools and files, so selecting a file also selects tools appropriate for that file.

How This Book Will Help You

The *Getting Started* guide will help you install and learn to use VisualAge COBOL. The book is divided into three sections. Each section provides important information for using VisualAge COBOL.

- The *Installing* section describes the installation process for VisualAge COBOL.
- The *Getting Started* section (“Getting Started with VisualAge COBOL” on page 35) provides an introduction to VisualAge COBOL. It includes product overview information, instructions for building two simple sample applications, and a quick-reference table for performing tasks using IBM VisualAge for COBOL for OS/2.
- The *Tutorial* section (“VisualAge COBOL Tutorials” on page 87) provides three, more complex tutorials to help you learn to use more features of VisualAge COBOL. In these tutorials, you will create a VisualAge COBOL application consisting of a main program and a subroutine, use the Data Assistant tool to map COBOL data structures to a DB2 database and generate SQL statements into your source file, and use the Transaction Assistant tool to generate a CICS ECI call and parameter list for invoking CICS transactions.

This book assumes familiarity with OS/2. For information on OS/2, see Appendix A, “Using OS/2” on page 131 or your operating system documentation.

Summary of Changes

January 1996—GC26-8421-01

Installation

This section contains updates to CONFIG.SYS entries and other changes.

Support Information

“Getting Support for Using VisualAge COBOL” on page 83 contains updated services and support information.

GUI COMPILE

Steps for the compile options for the Tax Computation Application have been updated in “Setting Compiler Options” on page 103.

Packaging Tool

The screens for the packaging tool have been updated in “Packaging the Application for Distribution” on page 113.

Data Assistant

Steps and screens in “Creating SQL Statements with Data Assistant” on page 117 have been updated.

Transaction Assistant

Minor changes have been made to the screens in “Using the CICS Transaction Assistant” on page 127.

VisualAge COBOL Sample Applications

Instructions on how to build and run the sample application have been added in Appendix B, “VisualAge COBOL Supplied Sample Applications” on page 139.

APPC Communications and IBM Debug Tool

Appendix C, “Configuring APPC Communications” on page 149 now contains information on how to configure the environment needed to debug host applications from the workstation using the IBM Debug Tool.

The online version of the *Debug Tool User's Guide and Reference* book is available from the **Information Notebook** in the main **VisualAge COBOL - Icon View** window.

Maintenance and editorial changes have been included throughout the book.

Installing VisualAge COBOL

The *Installing* section describes the installation process for IBM VisualAge for COBOL for OS/2, Version 1.

The topics in this section are:

Before You Begin	3
Installing IBM VisualAge for COBOL for OS/2	7
Adding or Deleting Components	19
If Something Goes Wrong	25
Response Files, Options, and Return Codes	29

Hardware and Software Requirements

Before You Begin

Before you begin to install IBM VisualAge for COBOL for OS/2, make sure that your workstation meets the hardware and software requirements described in "Hardware and Software Requirements."

Hardware and Software Requirements

Processor

Compiler only: an Intel** 80386- or 80486-based IBM workstation or equivalent.

Full product including visual COBOL development: an Intel 80486-based 66 MHz IBM workstation or equivalent.

Operating System

OS/2 V2.11 or OS/2 Version 3.0 (Warp).

OS/2 Version 3.0 (Warp) is not required to use the Warp Toolkit.

Note: Applications developed using VisualAge COBOL run on OS/2 V2.11 or later.

Other Software

To build applications requiring DB2 preprocessing and to use Data Assistant, you need IBM DB2 for OS/2 Version 2.1, including the Software Developer's Kit (SDK) as provided with the Single User Version.

To use Transaction Assistant, you need IBM CICS OS/2 Version 3.0 and CICS Client for OS/2 feature of CICS for OS/2 Version 2 or CICS Client for OS/2 Version 1.

To use the SMARTdata UTILITIES (SdU):

- To access MVS VSAM/SAM files, you need DFSMS Version 1.2.0 or later installed on your MVS host.
- To access MVS CICS-managed VSAM data, you need CICS DDM Release 1 installed on your MVS host.

To use host connectivity, you need Communications Manager/2 Version 1.11.

To use SdU and the remote edit and compile components, you need to configure your system as described in Appendix C, "Configuring APPC Communications" on page 149.

RAM

16 MB when using the Compiler only.

24 MB minimum for visual COBOL development.

28 MB recommended for visual COBOL development.

Hardware and Software Requirements

Disk Space

15 MB for the Compiler and Runtime component.

115 MB for the Compiler and Runtime component and COBOL tools.

155 MB for the Compiler and Runtime component, COBOL tools, OS/2 Toolkit, and Remote Edit/Compile component.

30 MB (minimum) for the swapper.

The program package contains COBOL beta code and optional components that are not needed to run COBOL. If you want to install them, you need additional disk space. You also need additional disk space for the swapper.

The tools and information are broken down into separate *components*. You can choose which components are to be installed. The Installation Utility displays the amount of disk space required for the components you selected. Table 1 shows the approximate disk space required for each component. If you are installing a corrective service disk (CSD), you may need additional disk space. See the *README.CSD* file located in the same directory as the CSD's install program.

Table 1 (Page 1 of 2). VisualAge COBOL Workstation Components Disk-Space Requirements

VisualAge COBOL Workstation Component	Disk Space Required (MB)
Compiler and Nonvisual Tools	
Compiler and Runtime	12
Information	8
Editor	7
Interactive Debugger	8
SMARTdata UTILITIES	4
Visual Tools	
COBOL GUI Designer	19
Data Assistant	6
Transaction Assistant	0.1
WorkFrame	12
Performance Analyzer	4
Data Description and Conversion	14
Warp Toolkit	
Development Tools	6
Information (not required for COBOL)	19
Headers and Libraries (not required for COBOL)	12
Sample Programs (not required for COBOL)	22
Multimedia Bitmaps (not required for COBOL)	0.1
Remote Edit/Compile	

Installing on LAN-Connected Workstations

Table 1 (Page 2 of 2). VisualAge COBOL Workstation Components Disk-Space Requirements

VisualAge COBOL Workstation Component	Disk Space Required (MB)
Remote Edit/Compile	7

Note: If you install to a FAT partition, the space required for each component may be larger than the amount shown. This is due to inefficiencies in the FAT file system. Also, if you install to the same disk that contains the swap file, allow additional space to compensate for swap file growth.

Note that you can install the components on different drives and directories.

Installing on LAN-Connected Workstations

If your workstations are connected to a LAN, you can set up 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 CD-ROM, and several client workstations can install at the same time.

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 can run the install program either attended or using a response file. For more information, see "Installing IBM VisualAge for COBOL for OS/2" on page 7.

Note: You **cannot** install VisualAge COBOL to a remote drive on the LAN.

If you are a LAN administrator:

Follow the steps below to put the appropriate installation files on the server.

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, COBOLCD).
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.

Installing on LAN-Connected Workstations

Installing IBM VisualAge for COBOL for OS/2

Installing IBM VisualAge for COBOL for OS/2

This section gives you step-by-step instructions for each of the VisualAge COBOL installation procedures. Make sure you have read "Before You Begin" on page 3.

If you get error messages during the installation procedure, select the **Help** push button in the message window for help on what to do. If you have problems installing, see "If Something Goes Wrong" on page 25.

You can install from a CD-ROM or from a LAN server. Check with your LAN administrator to find out if you can install from the server, and where the install image is located.

1. Insert the VisualAge COBOL CD-ROM, or access the LAN where the VisualAge COBOL image resides.
2. From the OS/2 command line, change to the directory where the install program is located. (For the CD-ROM, it is the root directory.)
3. Using the OS/2 system editor, read the *README* file located in the same directory as the install program.

This file contains the latest information about VisualAge COBOL changes or restrictions, including any that affect installation. If the *README* file differs from this install section, the *README* file is correct.

Before you install corrective service disks (CSD)

Be sure to read the *README.CSD* file located in the same directory as the CSD's install program.

4. At the command line type: `install`.

The main **VisualAge COBOL Installation** window and the **Instructions** window appear.

Installing IBM VisualAge for COBOL for OS/2

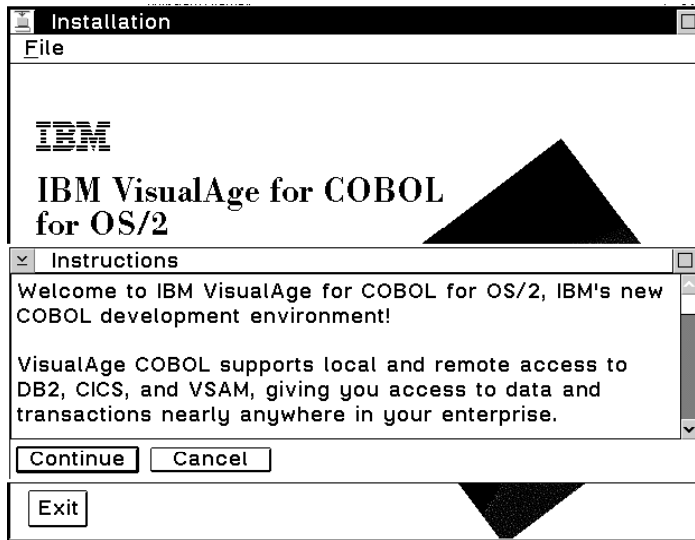


Figure 1. VisualAge COBOL Installation and Instructions windows

Scroll through the **Instructions** window to read all the text. You may need to move the **Instructions** window to see the main **Installation** window.

5. Select **Continue** in the Instructions window.

The **Install** window appears, displaying the product name and product and version numbers.

It also provides two checkboxes:

Update CONFIG.SYS

This box directs the install program to automatically update your CONFIG.SYS file. It also backs up CONFIG.SYS before making any changes. The backup name is CONFIG.001, or CONFIG.002 if that file exists, and so on. This box is checked by default. If you do not want the install program to update your CONFIG.SYS, remove the checkmark by deselecting the box.

Note: We recommend you leave this box checked to update CONFIG.SYS automatically. If you choose not to have the install program update your CONFIG.SYS, you must make the changes yourself before you run VisualAge COBOL and before you can complete the installation program for the WorkFrame component. For the changes, see the CONFIG.ADD file in the same drive and path as CONFIG.SYS. See "Changing Your CONFIG.SYS" on page 13 for information on what you need to change.

Overwrite files

Put a check in this box if you want to skip the prompt for confirmation when the install program overwrites files in your target directory with the same name as the files being installed.

6. When you have chosen the options you want, select **OK** to continue.

Installing IBM VisualAge for COBOL for OS/2

If you chose not to have the install program update CONFIG.SYS, a message box reminds you to update the file yourself when the installation is complete. Select **Yes** to continue.

The **Install - directories** window appears. From this window, you select the components to install and specify where to install them.

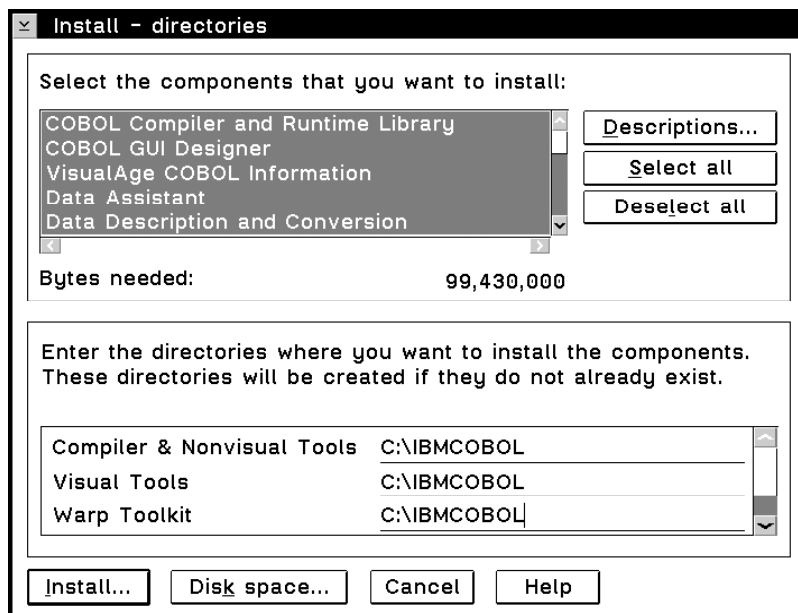


Figure 2. VisualAge COBOL Install-directories window

7. Choose the components you want to install from the list.

Components that are generally needed by a typical user are selected by default. Click on a component to deselect the ones you do not want, or use the **Deselect all** button to deselect all the components. You can then select the components you do want. To find out more about a component, select it (or multiple components), and then select **Descriptions**.

You do not have to install all the components at once; you can run the install program again at a later time (described in “Adding or Deleting Components” on page 19). You must select at least one component to continue with the install program.

Note: Some components require that you also install other components for them to work correctly. If you select a component that has such a prerequisite, a message will prompt you to also select the required component.

The following lists components that require other *components*:

- COBOL GUI Designer requires WorkFrame, Editor, Compiler and Runtime library, and Warp Toolkit Development Tools.

Installing IBM VisualAge for COBOL for OS/2

- WorkFrame requires the Warp Toolkit Development Tools.
- Data Assistant requires the Editor.
- Transaction Assistant requires the Editor.
- Remote Edit/Compile requires the Editor and WorkFrame.
- Data Description and Conversion requires SMARTdata UTILITIES.

As you choose the components, the **Bytes Needed** field shows you how much disk space is required to install them.

8. Specify the directory or directories where you want to install the components.

Default values are provided for you; they install everything to a single directory (IBMCOBOL) on the C: drive. You can install the components into different drives and directories. Make sure you specify the drive as well as the directory name.

If you name a directory that does not exist, the install program creates it for you.

To see how much disk space is available on your hard drive, select **Disk Space**. (You must have at least one component selected before you can select this button.) The Disk space window appears and shows you the space available on all drives, along with the bytes needed for the components you have chosen.

The install program does not prevent you from installing to a drive that does not have enough space. If you will be overwriting files, you may have enough space on a drive even though the installation program shows you do not. However, if there is not enough space, the installation for VisualAge COBOL will fail. See “If Something Goes Wrong” on page 25 for what to do if the program fails.

To set your installation directories to another drive, select the drive, check the **Change directories to selected drive** field, and select **OK**. The install program updates all of the directory fields to that drive, but does not change the directory names. To return to the main install window without making any changes, select **Cancel**.

9. When you have chosen the components to install and specified the target directories, select **Install** to start installing VisualAge COBOL.

The install program begins copying files to your hard drive. The **Install progress** window shows you the status of the installation.

If for any reason you want to stop the install program, click on **Stop** in the **Install progress** window. Once you have selected this button, you cannot resume the program. You must end it completely and start from the beginning. A window appears asking you whether you want to stop the installation. Select **Yes, stop** to close the window and end the install program. The install program also asks if you want to undo everything that it has done so far. We recommend you select **Yes** to undo everything. If you select **No**, the install program leaves all the files it has already installed on your hard drive. You will then need to erase these files before you restart the install program, or select **Overwrite files** in the **Install** window.

10. If you selected the Remote Edit/Compile component, the license agreement for the Remote Edit/Compile component window appears asking you if you agree with the terms and conditions. Click **Yes** or **No**.

Installing IBM VisualAge for COBOL for OS/2

11. When the installation is nearly complete (and you chose to install the SMARTdata UTILITIES component), the **SMARTsort Installation** prompt appears. You will be setting some options to control the default behavior of SMARTsort. Select **OK**.

The **SMARTsort Defaults** window appears. The **Default Memory** setting determines how much memory SMARTsort will attempt to allocate during the sorting process. The default working directory shown in the **Default Directories** field stores the temporary work files created. For more information, select **Help**, or see the **SdU Library** icon in the **Books** folder from the main **VisualAge COBOL - Icon View** window.

When you are done, click **OK**.

12. If you selected the Compiler and Runtime library component, a message dialog appears asking you whether or not you want to change the compiler default options. If you click **OK**, the **COBOL Compiler Default Options Tool** dialog screen appears as shown in Figure 3. This dialog allows you to accept or change the compiler default options. For more information about the compiler options, click on the **Help** push button, or see *IBM VisualAge for COBOL for OS/2 Programming Guide*.

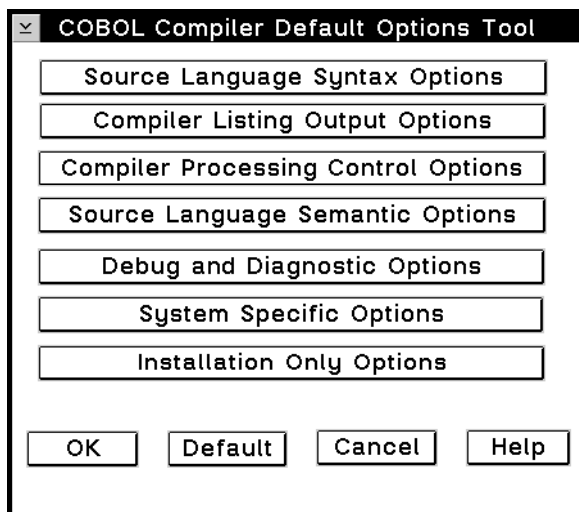


Figure 3. COBOL Compiler Default Options Tool

Note: Use the tool to set the default value of the compiler options when they are 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**.

Installing IBM VisualAge for COBOL for OS/2

- 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**.

After the installation is complete:

- When you invoke the tool, it reads the *current* default options, which may not be the default options provided by IBM.
 - To use the tool, you must have write access to the directory where the COBOL compiler is installed.
 - You can invoke the tool at any time. Change to the DLL subdirectory where you installed the COBOL compiler. For example, type:

```
cd IBMCOBOL\DLL
DIAMOND
```
 - You can invoke the tool by double-clicking on the **Compiler Default Options Tool** icon in the **VisualAge COBOL Works** folder.
13. A message window tells you when the installation is complete. Select **OK** to remove the message window and return to the main VisualAge COBOL install window.
 14. In the main install window, select the **Exit** button, or **Exit** from the **File** menu to end the install program.
Note: If you chose not to have the installation program update your CONFIG.SYS, make the changes to it now **before** you reboot (see “Changing Your CONFIG.SYS” on page 13).
 15. Shut down and restart your system to make the changes to your CONFIG.SYS file take effect.
 16. If you chose to install the WorkFrame component, the installation program requires a second phase, which is invoked from your **Startup** folder when you restart. During the second phase, the installation program completes the WorkFrame installation process.

An OS/2 window with the title **WorkFrame V3 Phase 2 Install** appears after you have restarted. Messages appear in the window to inform you of the install program's actions. When the installation is complete, the window disappears.

If you want to verify that the installation completed successfully, look for IWZINST.LOG in your VisualAge COBOL directory (Visual Tools). The second phase of the install program creates this file and timestamps the entries in it.

You have now successfully installed VisualAge COBOL, and are ready to go! See “Getting Started with VisualAge COBOL” on page 35 for what to do next.

Changing Your CONFIG.SYS

Changing Your CONFIG.SYS

We recommend that you let the Installation Utility update your CONFIG.SYS file automatically. However, if you chose not to have the install program update your CONFIG.SYS file, you must make changes to the statements it contains before you can use VisualAge COBOL. If you are installing WorkFrame, you must make these changes before the second phase of the installation can begin.

The Installation Utility creates a file called CONFIG.ADD, which is a copy of your CONFIG.SYS file *with* the VisualAge COBOL changes.

When you have finished changing CONFIG.SYS, restart your system to make the changes take effect.

The following lists the component and its environment variables for CONFIG.SYS.

- Directories are indicated as follows. Table 1 on page 4 lists the specific components for each directory (Compiler and Nonvisual Tools, Visual Tools, Remote Edit/Compile, and Warp Toolkit).

<nonvis>

indicates the Compiler and Nonvisual Tools directory.

<vis>

indicates the Visual Tools directory.

<remote>

indicates the Remote Edit/Compile directory.

<toolkit>

indicates the Warp Toolkit directory.

- Ellipses (...) indicate that your other values will remain the same.
- All variable values are placed at the beginning of the statements, rather than at the end.

COBOL Compiler & Runtime Library

```
LIBPATH=<nonvis>\dll...
SET PATH=<nonvis>\bin...
SET DPATH=<nonvis>\messages;<nonvis>\help...
SET TMP=<vis>\tmp
SET COBPATH=<nonvis>\dll
SET LIB=<nonvis>\lib...
SET NLSPATH=<nonvis>\messages\enus437\%N...
SET LOCPATH= <nonvis>\locale...
SET LANG= enus437
SET ICOBDIR=<vis>
SET ICOBFDIR=<nonvis>
SET HELP=<vis>\help...
```

Changing Your CONFIG.SYS

COBOL GUI Designer

```
LIBPATH=.;<vis>\dll;<vis>\ctrl...
SET PATH=.;<vis>\bin...
SET DPATH=<vis>\messages;<vis>\macros...
SET COBPATH=<nonvis>\dll
SET LIB=<vis>\lib...
SET HELP=<vis>\help...
SET BOOKSHELF=<vis>\help...
SET IWF.SOLUTION_LANG_SUPPORT=iwzv\logo;eng
```

VisualAge COBOL Information

```
LIBPATH=<nonvis>\dll...
SET PATH=<nonvis>\bin...
SET HELP=<nonvis>\help...
SET BOOKSHELF=<nonvis>\help...
```

Data Assistant

```
LIBPATH=<vis>\dll...
SET PATH=<vis>\bin...
SET DPATH=<vis>\icon;<vis>\macros...
SET HELP=<vis>\help...
SET ICOBDIR=<vis>
```

Editor

```
LIBPATH=<nonvis>\dll...
SET PATH=<nonvis>\bin...
SET DPATH=<nonvis>\macros;<nonvis>\help...
SET HELP=<nonvis>\help...
SET BOOKSHELF=<nonvis>\help...
SET LPATH=<nonvis>\macros...
```

Interactive Debugger

```
LIBPATH=<nonvis>\dll...
SET PATH=<nonvis>\bin...
SET DPATH=<nonvis>\messages...
SET HELP=<nonvis>\help...
```

Performance Analyzer

```
LIBPATH=<vis>\dll...
SET PATH=<vis>\bin...
SET DPATH=<vis>\help...
SET LIB=<vis>\lib...
SET HELP=<vis>\help...
SET BOOKSHELF=<vis>\help...
DEVICE=<vis>\sys\cppopa3.sys
```

Remote Edit/Compile

Changing Your CONFIG.SYS

```
LIBPATH=<remote>\dll...
SET PATH=<remote>\bin...
SET DPATH=<remote>\macros...
SET BOOKSHELF=<remote>\help...
SET HELP=<remote>\help...
SET CODESHRDIR=<remote>\code
SET CODETMPDIR=<vis>\tmp
SET CODELPATH=<remote>\macros;<remote>\macros\code
SET CODEHOSTCP=037
SET CODEINIDIR=<remote>\code
SET IWFPAM=iwfbpam iwfpms
```

SMARTdata UTILITIES

```
LIBPATH=<nonvis>\dll...
SET PATH=<nonvis>\bin...
SET DPATH=<nonvis>\bin\mri2924...
SET INCLUDE=<nonvis>\include...
SET LIB=<nonvis>\lib;<vis>\lib...
SET EHNL=2924
SET EHNDIR=<nonvis>\bin
IFS=<vis>\dll\dfmsf10.dll
SET NLSPATH=<vis>\messages\enus437\%N...
SET LOCPATH=<nonvis>\locale...
SET LANG=enus437
```

Data Description and Conversion

```
LIBPATH=<nonvis>\dll...
SET PATH=<nonvis>\bin...
SET INCLUDE=<nonvis>\include...
SET LIB=<nonvis>\lib;<vis>\lib...
SET FMTDIR=<nonvis>
SET FMTCBRA=<nonvis>\bin\convtab1
```

Transaction Assistant

```
LIBPATH=<vis>\dll...
SET LIB=<vis>\lib...
SET HELP=<vis>\help...
```

WorkFrame

```
LIBPATH=<vis>\dll...
SET PATH=<vis>\bin...
SET DPATH=<vis>\help...
SET HELP=<vis>\help...
SET BOOKSHELF=<vis>\help...
SET IWFPAM=iwfbpam
SET IWF.SOLUTION_LANG_SUPPORT=iwzvlogo;eng
```

Installing Using Response Files

Warp Toolkit Development Tools

```
LIBPATH=<toolkit>\dll...
SET PATH=<toolkit>\bin...
SET LIB=<toolkit>\lib...
SET HELP=<toolkit>\help...
SET BOOKSHELF=<toolkit>\help...
SET DPATH=<toolkit>\help...
SET IPFC=<toolkit>\ipfc
SET SOMIR=<toolkit>\etc\som.ir...
SET SOMRUNTIME=<nonvis>\som
```

Warp Toolkit Information

```
SET HELP=<toolkit>\help...
SET BOOKSHELF=<toolkit>\help...
SET CPREF=cp1.inf+cp2.inf+cp3.inf
SET GPIREF=gpi1.inf+gpi2.inf+gpi3.inf
SET PMREF=pm1.inf+pm2.inf+pm3.inf+pm4.inf+pm5.inf
SET WPSREF=wps1.inf+wps2.inf+wps3.inf
SET MMREF=mmref1.inf+mmref2.inf+mmref3.inf
```

Warp Toolkit Headers & Libraries

```
LIBPATH=<toolkit>\dll;<vis>\dll...
SET INCLUDE=<toolkit>\include;<toolkit>\include\os2;<toolkit>\inc...
SET SOMRUNTIME=<nonvis>\som
SET SMINCLUDE=<toolkit>\include;<toolkit>\include\os2...
SET SMADDSTAR=1
SET SMEMIT=h;ih;c
SET SOMBASE=c:\ibmcobol
SET SMTMP=<vis>\tmp
SET SMCLASSES=wptypes.idl
```

Warp Toolkit Sample Programs

```
LIBPATH=<toolkit>\dll;<vis>\dll;<toolkit>\samples\toolkit\dll...
SET HELP=<toolkit>\samples\toolkit\help...
```

Warp Toolkit Multimedia Bitmaps

No environment variables are required.

When you have finished changing CONFIG.SYS, restart your system to make the changes take effect.

Installing Using Response Files

If you are installing from a CD-ROM or LAN server, you can run the install program unattended, using a response file to specify what to install and where.

To install using a response file:

1. Create a response file, or tailor the sample response file, UNATTEND.RSP, provided in the IBMCOBOL\EXTRAS directory of the CD-ROM. You can copy

Installing Using Response Files

UNATTEND.RSP to your hard drive and change the appropriate responses. Make sure that:

- FILE specifies the directory where you want to install the Compiler and Non-visual Tools.
- WORK specifies the directory where you want to install the Visual Tools.
- AUX1 specifies the directory where you want to install the Warp Toolkit.
- AUX2 specifies the directory where you want to install the Remote Edit/Compile component.
- COMP keywords specify the components you want to install.

For a list of all the parameters you can change, see “Response Files, Options, and Return Codes” on page 29.

2. From an OS/2 command line, change to the CD-ROM or LAN directory where the install program resides.
3. Invoke the install program with the command:

```
install /A:I /X /R:d:UNATTEND.RSP /C:IBMCOBOL.ICF /P:"IBM VisualAge for COBOL for OS/2" /O:DRIVE
```

where d:UNATTEND.RSP is your response file.

Note: If you want to log any error messages that are generated, specify the `/L1:errorlog` option, where *errorlog* is the path and file name to use for the error log. This is usually a good idea because messages are not displayed during an unattended install; if you don't log them, you will have no record of what errors occurred.

For a description of these and other command-line options, see “Response Files, Options, and Return Codes” on page 29.

4. You can then leave the install program unattended. When it is complete, the OS/2 command prompt appears in the OS/2 session where you ran the `install` command.
5. If you chose not to have the installation program update your CONFIG.SYS, make the changes to it now **before** you reboot. (See “Changing Your CONFIG.SYS” on page 13 for details on the changes to make).
6. Shut down and restart your system to make the changes to your CONFIG.SYS file take effect.
7. If you chose to install the WorkFrame component, the installation program requires a second phase, which is invoked from your **Startup** folder when you restart. During the second phase, the install program completes the WorkFrame installation process.

An OS/2 window with the title **WorkFrame V3 Phase 2 Install** appears after you have restarted. Messages appear in the window to inform you of the install program's actions. When the installation is complete, the window disappears.

If you want to verify that the installation completed successfully, look for IWZINST.LOG in your VisualAge COBOL directory (Visual Tools). The second phase of the install program creates this file and timestamps the entries in it.

Using a Software Distribution Manager

You have now successfully installed VisualAge COBOL, and are ready to go! See “Getting Started with VisualAge COBOL” on page 35 for what to do next.

Using a Software Distribution Manager

You can also install VisualAge COBOL unattended over a LAN using a software distribution manager (SDM). Tailor the response file according to your needs and use it with your SDM. Make sure you specify the correct options, as described above.

Your SDM must also check the value returned by the install program and perform the appropriate action. Return codes and the actions required for each are described in “Response Files, Options, and Return Codes” on page 29, along with all response file parameters and command-line options.

Adding or Deleting Components

Adding or Deleting Components

Once VisualAge COBOL has been installed, you can go back to the install program to install additional components or to delete components. If you need to reinstall a component, you can delete it and then add it again.

You can use one of three ways to add or delete components:

1. Run the original install program interactively.
2. Use the Installation Utility icon from the VisualAge COBOL **Works** folder.
3. Run the original install program with a response file.

The steps for methods 1 and 2 are very similar, other than the initial interface. The steps for using a response file are described in "Adding or Deleting Components Using a Response File" on page 23.

If you are adding components, make sure you have access to the CD-ROM or LAN directory where the VisualAge COBOL installation files reside.

To add or delete components with the original install program:

1. Insert the VisualAge COBOL CD-ROM, or access the LAN where the VisualAge COBOL image resides.
2. From the command line, change to the directory where the install program is located. (For the CD-ROM, it is the root directory; for the LAN directory name, ask your LAN administrator.)
3. On the command line, type `install`.

The main **VisualAge COBOL installation** window and the **Instructions** window appear.

4. Select **Continue** in the Instructions window.

The **Installation options** window appears showing three options.

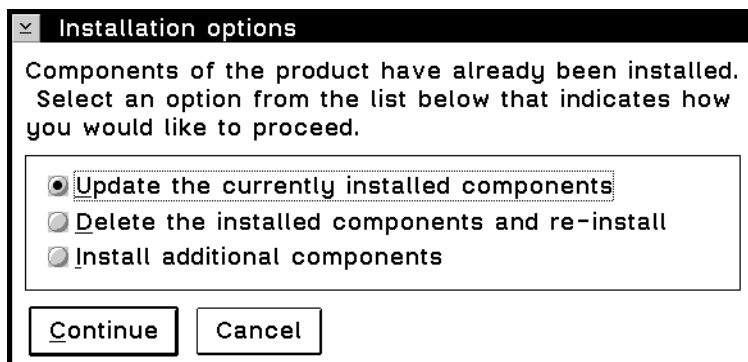


Figure 4. Installation options window

Deleting Components

The options are:

- **Update the currently installed components**

Do not use this option. It is provided to support corrective service (CSDs) for VisualAge COBOL fixes. To reinstall a component, delete it first, then reinstall.

- **Delete the installed components and re-install**

Choose this option to delete the installed components, including their Desktop objects and entries in any .INI files. You can then reinstall them if you want. The steps for deleting components continue in “Deleting Components.”

- **Install additional components**

Choose this option to install components that are not already installed. The steps for adding components continue in “Installing Additional Components” on page 22.

To add or delete components using the Installation Utility:

1. Open the **Works** folder inside the **VisualAge COBOL - Icon View** folder, or from the command line, go to the VisualAge COBOL directory (Compiler and Nonvisual Tools).
2. Double-click on the **Installation Utility** icon, or enter `epfinsts` from the command line.

The **Installation Utility** window appears.

3. If you are adding components, you must select **Open Catalog** from the **File** pull-down. Select **Drive**. The **Open drive catalog** window appears. In the **Drive** entry field, select the drive where the install image resides (CD-ROM or a LAN drive). In the **Filename** entry field, ensure that `/IBMCOBOL.ICF` is displayed. Click on **Open**.
4. From the **Action** pull-down, select:
 - **Delete** to delete installed components, including their Desktop objects and any entries in .INI files. You can then reinstall them if you want. The steps for deleting components continue in “Deleting Components.”
 - **Install** to install additional components. The steps for adding components continue in “Installing Additional Components” on page 22.

Note: Do not choose **Update**. This option is provided for corrective service diskettes (CSDs) that contain product fixes. The **Restore** option restores a backup version of a component, which you can choose to create when you update the component with a CSD.

Deleting Components

When you choose **Delete the installed components and re-install** from the Installation options window, or **Delete** from the **Action** pull-down of the Installation Utility:

1. The Delete window displays a list of the components you have installed.

Deleting Components

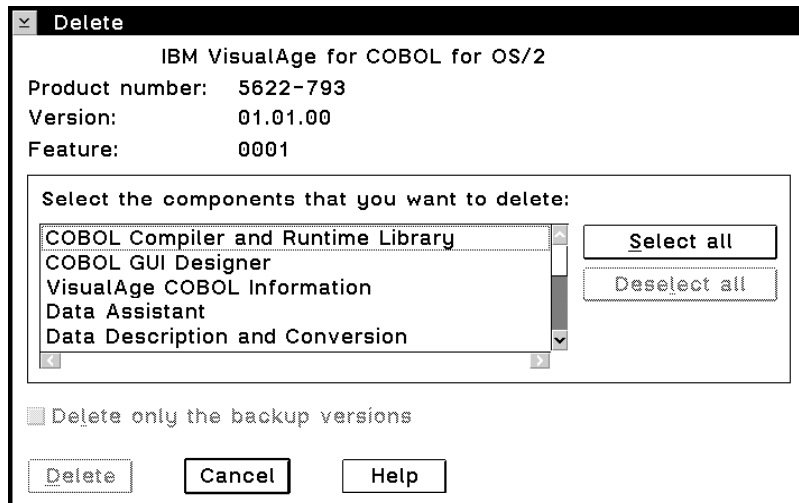


Figure 5. Delete window

2. Select the components you want to delete.

You can use the **Select all** button to choose all the components, or **Deselect all** to deselect them and start again.

If you previously applied fixes to components and created backup versions, you can delete just the backup versions by selecting the components and checking the **Delete backup versions only** box.

3. When you have chosen the components to delete, select **Delete**.

The **Delete - progress** window shows the status of the deletion process. A message window tells you when all of the component files and related information have been deleted.

4. Select **OK** to remove the message window.

Notes:

- a. You may get a message indicating that some files could not be deleted because they were in use. Select **OK** to continue. When you shut down and restart your system, these files will be deleted.
- b. If you get this message, it is important to shut down and restart your system before installing any components.

If you used the Installation Utility to delete the files, it reappears. If you used the install program (install), the Installation options window reappears, as shown on page 19.

5. To reinstall components, select **Install additional components** from the Installation options window, or select **Install** from the Installation Utility's **Action** pull-down.

Installing Additional Components

To end the install program, select the **Exit** button in the main install window, or **Exit** from the **File** menu. To end the Installation Utility program, select **Exit** from the **File** menu.

Installing Additional Components

When you choose **Install additional components** from the Installation options window, or **Install** from the **Action** pull-down of the Installation Utility:

1. The Install window appears, as it did in the initial installation procedure, with two checkboxes:

Update CONFIG.SYS

This box directs the install program to automatically update your CONFIG.SYS file. (It also backs up CONFIG.SYS before making any changes.) This box is checked by default; if you do not want the install program to update your CONFIG.SYS, remove the checkmark by deselecting the box.

Note: If you choose not to have the install program update your CONFIG.SYS, you must make the changes yourself before you reboot. A CONFIG.ADD file is created on the C: drive to help you. See “Changing Your CONFIG.SYS” on page 13 for information on what you need to change.

Overwrite files

Put a check in this box if you want to skip the prompt for confirmation when the install program overwrites files in your target directory with the same name as the files being installed.

Note: The installation program does not delete any obsolete files from earlier versions.

2. When you have chosen the options you want, select **OK** to continue.

If you chose not to have the install program update CONFIG.SYS, a message box reminds you to update the file yourself. Select **Yes** to continue with the installation.

The **Install - directories** window appears, looking slightly different than it did in the initial installation.

Only components that you have not yet installed are displayed.

3. Select the components you want to install from the list.

As you choose the components, the **Bytes Needed** field shows you how much disk space is required to install them. To get the description of a component, select the component and select **Descriptions**. To see how much disk space is available on your hard drive, select **Disk Space**. (You must have at least one component selected before you can select this button.)

Any components that you choose are installed to the directories that were created during the initial installation. If you want to install to different directories, you must delete the original components and reinstall them to a different directory.

Adding or Deleting Components Using a Response File

4. When you have chosen the components to install, select **Install** to start installing VisualAge COBOL.

The **Install progress** window shows you the status of the installation.

5. A message window tells you when the installation is complete. Select **OK** to remove the message window and return to the Installation Utility window or the main VisualAge COBOL install window where you started from.
6. To end the Installation Utility program, select **Exit** from the **File** menu. To end the install program, select the **Exit** button, or **Exit** from the **File** menu.

Note: If you chose not to have the installation program update your CONFIG.SYS, make the changes to it now **before** you reboot (see “Changing Your CONFIG.SYS” on page 13).

7. Shut down and restart your system to make the changes to your CONFIG.SYS file take effect.
8. If you chose to install the WorkFrame component, the install program requires a second phase, which is invoked from your **Startup** folder when you restart. During the second phase, the install program registers WorkFrame classes, updates any sample projects you installed, and sets up COBOL Project Smarts.

An OS/2 window with the title **WorkFrame V3 Phase 2 Install** appears after you have restarted. Messages appear in the window to inform you of the install program's actions. When the installation is complete, the window disappears.

If you want to verify that the installation completed successfully, look for IWZINST.LOG in your VisualAge COBOL directory (Visual Tools). The second phase of the install program creates this file and timestamps the entries in it.

Adding or Deleting Components Using a Response File

To add or delete components using a response file:

1. Modify the appropriate parameters in your response file.

If you are deleting components, make sure that:

- COMP keywords specify the components to delete.
- DELETEDBACKUP specifies whether you want to delete backup files,

If you are adding components, make sure that:

- FILE specifies the directory where you want to install the Compiler and Non-visual Tools.
- WORK specifies the directory where you want to install the Visual Tools.
- AUX1 specifies the directory where you want to install the Warp Toolkit.
- AUX2 specifies the directory where you want to install the Remote Edit/Compile component.
- COMP keywords specify the components you want to install.

2. Change to the CD-ROM or LAN directory where the VisualAge COBOL install program resides.

Adding or Deleting Components Using a Response File

3. To delete components, enter:

```
install /A:D /X /R:d:UNATTEND.RSP /C:IBMCOBOL.ICF /P:"IBM VisualAge for COBOL for OS/2" /O:DRIVE
```

where *d*:UNATTEND.RSP is your response file.

To install additional components, enter:

```
install /A:I /X /R:d:UNATTEND.RSP /C:IBMCOBOL.ICF /P:"IBM VisualAge for COBOL for OS/2" /O:DRIVE
```

4. You can then leave the install program unattended while the components are deleted or installed.
5. When the install program has finished, the OS/2 command prompt appears in the OS/2 session where you ran the `install` command.
6. If you added components and chose not to have the installation program update your CONFIG.SYS, make the changes to it now **before** you reboot. (See "Changing Your CONFIG.SYS" on page 13 for details on the changes to make).
7. Shut down and restart your machine.
8. If you added the WorkFrame component, the install program requires a second phase, which is invoked from your **Startup** folder when you restart. During the second phase, the install program registers WorkFrame classes, updates any sample projects you installed, and sets up COBOL Project Smarts.

An OS/2 window with the title **WorkFrame V3 Phase 2 Install** appears after you have restarted. Messages appear in the window to inform you of the install program's actions. When the installation is complete, the window disappears.

If you want to verify that the installation completed successfully, look for IWZINST.LOG in your VisualAge COBOL directory (Visual Tools). The second phase of the install program creates this file and timestamps the entries in it.

If Something Goes Wrong

If Something Goes Wrong

This section describes what to do if you encounter a problem or an error when you run the install program. Online help is also available for error messages, both from the **Help** button on the message window, or from the index or contents of the online install help.

If You Get an Error Message

Select the **Help** button for the error for information on how to correct it. Also note the message number and the file name where the error occurred, if one is indicated. If you cannot successfully complete the installation and have to call VisualAge COBOL Service and Support, this information may help identify your problem (see "Getting Support for Using VisualAge COBOL" on page 83).

If the Component Requires Another Component

Many of the VisualAge COBOL components require that other components be installed for them to work correctly. If you select a component to install without selecting its prerequisite, a message prompts you to select the prerequisite also.

Select the prerequisite component, as indicated by the message. Note that the prerequisite component may itself have prerequisites.

If You Press Stop

If you press the **Stop** button on the Install progress window, the install program immediately stops and displays a window that asks if you want to back out of the installation so far. You cannot resume the install program from this point; you must end it and start again.

In the confirmation window:

- Select **Yes** to undo everything that the install program has done up to that point. We recommend you choose this option. Any desktop objects are deleted.
- Select **No** to leave the files that have already been installed.

Once the program has ended, you can restart it from the beginning.

If the Install Program Fails

If the install program fails, it displays a window that asks if you want to back out of the installation:

- Select **Yes** to undo everything that the install program has done up to that point. We recommend you choose this option.
- Select **No** to leave the files that have already been installed.

If Something Goes Wrong

The most likely reason for the failure is that not enough disk space is available. To solve this problem:

- Make more room on your target drive.
- Choose a different target drive.
- Select fewer components to install.

If disk space is not a problem, run CHKDSK on the:

- Drive where OS/2 is installed
- Target drive to ensure there are no file system problems.

If you are using a response file, make sure you specified responses for AUX1, AUX2, CFGUPDATE, DELETEBACKUP, FILE, OVERWRITE, SAVEBACKUP, and WORK, and at least one component name for COMP. If you do not give responses for these keywords, the install program may fail. Make sure that you specified the component names correctly. Also make sure that you specified the required command-line options as described in “Installing Using Response Files” on page 16.

If you try the install program again with the response file, specify the command-line options `/L1:errorlog` and `/L2:historylog` to create an error log and history log, respectively. Specify the path and file name for both. If the problem still occurs, these logs may help you determine the cause.

If you have tried all of these suggestions and the install program still fails, contact VisualAge COBOL Service and Support (see “Getting Support for Using VisualAge COBOL” on page 83).

If the Second Phase of Installation Fails

If the second phase of the install program fails, try running it again:

1. From an OS/2 command line, change to the Visual Tools directory where you installed VisualAge COBOL.
2. Type: `iwzvfaz2`

This command displays the correct parameters you need to provide to the Phase 2 install program.

You also need to remove the WorkFrame Phase 2 install icon from your **Startup** folder. (This icon is removed automatically if the initial phase 2 install works correctly.) If you do not remove it, the Phase 2 install will run every time you restart your system.

If you want to verify that the installation completed successfully, look for IWZINST.LOG in your VisualAge COBOL directory (Visual Tools). The second phase of the install program creates this file and timestamps the entries in it. Look for any entries containing the word *failed*.

If Something Goes Wrong

If the Second Phase of the Installation Does Not Run

The install program causes the Phase 2 install to run by placing a program object in your Startup folder. You may have added SET RESTARTOBJECTS=NO in your CONFIG.SYS file. This prevents programs in your Startup folder from running automatically. You can manually run the Phase 2 install by double-clicking its program object in your Startup folder.

If You Can't Delete an Object Because It's in Use

When you delete a component, if any objects or files are in use, they are not deleted. A message informs you when this happens. The rest of the component and its desktop objects are deleted.

After you shut down and restart your system, these objects should be deleted automatically.

If they are not, run the delete action again by changing to the Compiler and Nonvisual Tools directory, where you installed VisualAge COBOL. From an OS/2 command line, go to where the Compiler and Nonvisual Tools directory is located and type `epfnsts`.

If VisualAge COBOL Doesn't Appear in the Installation Utility

If you start the Installation Utility and VisualAge COBOL is not listed in the window, select **Current Catalog** from the **View** menu to display it.

If You Can't Install Additional Components

If you attempt to use the Installation Utility to install additional components and get an error of EPFIE114, you must select the drive where the install image resides. See step 3 on page 20.

If You've Tried Everything and It Still Doesn't Work

If the install program continues to fail after you've tried everything suggested by the error message help and this section, contact VisualAge COBOL Service and Support (see "Getting Support for Using VisualAge COBOL" on page 83).

Be sure to inform them what error messages you see, and what file names, if any, are given in those messages. If you are using response files, tell them the contents of your error and history logs.

If Something Goes Wrong

Response Files, Options, and Return Codes

Response Files, Options, and Return Codes

This section describes all the response file parameters that you can tailor for an unattended install, the different command-line options you can use, and the values the install program returns.

Response File Format and Parameters

The response file is a flat ASCII file that consists of a number of response lines and optional comment lines. Lines can be up to 255 bytes in length, and are separated by a new-line sequence.

A comment line begins with an asterisk (*) or semi-colon (;). Response lines tell the install program how to install. Each response line has the format:

keyword = response

Keywords are not case sensitive.

For the VisualAge COBOL install program, the keywords and the responses to specify for each are:

Keyword	Response
AUX1	The directory where you want to install the Warp Toolkit.
AUX2	The directory where you want to install the Remote Edit/Compile component.
CFGUPDATE	AUTO to automatically update CONFIG.SYS; MANUAL to not update CONFIG.SYS. If you choose not to update CONFIG.SYS automatically, you must update it yourself before you reboot (see "Changing Your CONFIG.SYS" on page 13). You will also be prompted by the install program to confirm that this is what you want.
COMP	The name of a component to install. You must specify each component with its own COMP keyword. Note that many components require that other components be installed to work correctly. All the components are listed in UNATTEND.RSP; you can delete any you do not want.
COPY	The source and target files for a copy process outside of the install program. This parameter is useful if you are using an SDM to install VisualAge COBOL over a LAN, and you want to copy other files that are not part of VisualAge COBOL to the workstations. The format for COPY is:

COPY = sourcefile targetfile

If *targetfile* already exists, it is overwritten. If either file specification is incorrect, the copy is not done.

Response Files, Options, and Return Codes

DELETEBACKUP

YES to delete a backup version along with the product when Delete is chosen; NO to keep the backup version. This setting is only used when you delete the product. See “Adding or Deleting Components” on page 19 for details on deleting the product.

Note: If you do not specify this keyword for an unattended delete, the deletion fails.

FILE

The directory where you want to install the Compiler and Nonvisual Tools.

INCLUDE

The name of another response file to include. You can have up to five levels of nested response files. If you don't specify the fully-qualified file name, the install program looks for the response file using the following search order:

1. The current directory.
2. The path specified by the /G command-line option.
3. Directories specified by the PATH environment variable.
4. Directories specified by the DPATH environment variable.

If the file name specified contains a wildcard character (* or ?), the first matching file is used.

OVERWRITE

YES to overwrite files on the target install drive; NO if you do not want to overwrite files. Note that if you specify NO and files exist on that drive with the same name as VisualAge COBOL files, the installation will not continue.

SAVEBACKUP

YES to save a backup copy; NO to not save a backup. This setting is only used when you install corrective service (CSDs) for VisualAge COBOL.

USEREXIT

The name of a program that you want the install program to call. This is useful if you are installing VisualAge COBOL using an SDM, and want to perform additional tasks. If you do not specify the fully-qualified file name, the install program looks for the program file using the following search order:

1. The current directory.
2. Directories specified by the PATH environment variable.
3. Directories specified by the DPATH environment variable.

If the file name specified contains a wildcard character (* or ?), the first matching file is used.

WORK

The directory where you want to install the Visual Tools.

You must specify responses for the AUX1, AUX2, CFGUPDATE, DELETEBACKUP, FILE, OVERWRITE, SAVEBACKUP, and WORK keywords, and specify at least one component for the COMP keyword, for the install program to work correctly.

Command-Line Options

Command-Line Options

You can specify a number of command-line options for the `install` command:

`/A`
`/C`
`/O`
`/P`
`/R`
`/X`

All other options are optional.

The command-line options are:

`/A:action`

Specifies the action to perform. *action* can be any of:

- D** Delete.
- I** Install.
- R** Restore.
- U** Update. (Note that you should only use this action when you are installing corrective service (CSDs) to VisualAge COBOL.)

`/C:d:\dir\c:IBMCOBOL.ICF`

Specifies the catalog file that contains the information about the VisualAge COBOL files. You must specify the drive and directory, which are the same as the install program.

`/G:includepath`

Specifies the path the install program should use to locate response files.

`/L1:d:\dir\errorlog`

Specifies the error log file. The install program logs any errors to this file and prefixes them with a timestamp. If you do not specify the path and file name, IWZINSTS.OUT is created in the temporary install directory (usually on the drive with the most available space). If you do not specify this option, messages are not logged.

`/L2:d:\dir\historylog`

Specifies the history log file. The install program logs the install events in this file and prefixes them with a timestamp. If you do not specify a path and file name, the history log is created in the temporary install directory (usually on the drive with the most available space). If you do not specify this option, the history is not logged.

`/O:DRIVE`

Specifies that the program files are being copied from a local or remote disk drive, not a mainframe host.

`/P:"IBM VisualAge for COBOL for OS/2"`

Specifies the name of the product to install.

Return Codes

/R:d:\dir\UNATTEND.RSP

Specifies the response file to use. If you do not specify the fully-qualified file name, the install program looks for the response file using the following search order:

1. The current directory.
2. The path specified by /G, if any.
3. Directories specified by the PATH environment variable.
4. Directories specified by the DPATH environment variable.

/S:d:\sourcedir

Specifies the directory where the source files reside.

/T:d:\targetdir

Specifies the directory where the files should be installed. If you use this option, it overrides what is specified for FILE in the response file.

/TU:d:\dir\CONFIG.SYS

Specifies the CONFIG.SYS file to be updated.

/X Specifies that the install program runs unattended, using a response file.

Return Codes

The install program returns a 2-byte hexadecimal value to the SDM, indicating success or failure, what steps should be taken next, and what type of messages, if any, were logged:

- Successful installation. No other action required.
00 00 No messages were logged.
- Successful installation. Restart the workstation operating system. Do not call the install program again.
FE 00 No messages were logged.
FE 04 Warning messages were logged.
FE 08 Error messages were logged.
FE 12 Severe error messages were logged.
- Successful installation. Restart the workstation operating system and call the install program again.
FF xx xx can be any value from 00 to FF.
- Installation did not complete successfully; an expected condition was encountered.
16 00 The install program was invoked incorrectly.
16 04 Messages were logged.

Return Codes

If you created user exits for the install program to call (as specified by the USEREXIT keyword in the response file), your user exit must return a 2-byte hexadecimal value to the install program as follows:

- 00 00** Your program completed successfully.
- FE 00** Your program completed successfully, and requires the workstation operating system to be restarted without calling the install program again. When you return this value, the install program displays a message to restart the operating system.
- FF xx** Your program completed successfully, and requires the workstation operating system to be restarted and the install program to be called again. When you return this value, the install program displays a message to restart the operating system and to try the action again.

If your program is in REXX, you can use the REXX EXIT command and return the value in decimal instead of hexadecimal.

If your program returns a different value to the install program, the install program displays a message that a product-specific error occurred, indicating the name of your program and the return code.

Return Codes

Getting Started with VisualAge COBOL

The *Getting Started* section provides an introduction to the VisualAge COBOL environment.

“Getting Started with VisualAge COBOL” assumes that you have installed the product on your workstation, and that you are ready to get going. If you haven’t installed the product yet, follow the instructions in “Installing VisualAge COBOL” on page 1 of this book.

You also need to know how to use a mouse to manipulate windows, icons, and other objects on the OS/2 desktop. Appendix A, “Using OS/2” on page 131 presents a brief overview of using OS/2. For more practice using OS/2, you can take the OS/2 Tutorial. The OS/2 Tutorial provides a hands-on overview of basic tasks. It is normally located in the Information folder on your desktop.

The topics in this section are:

Introducing IBM VisualAge for COBOL for OS/2	37
Build Your First VisualAge COBOL Application	43
Build Your First VisualAge COBOL GUI Application	49
Tools in VisualAge COBOL	69
Tasks and Information for VisualAge COBOL	75
Using the Information with VisualAge COBOL	79
Getting Support for Using VisualAge COBOL	83

Introducing VisualAge COBOL

Introducing IBM VisualAge for COBOL for OS/2

IBM VisualAge for COBOL for OS/2 is a COBOL development environment for creating applications on OS/2.

It offers the best of both traditional and cutting-edge COBOL programming. It provides a set of workstation tools for developing COBOL applications. VisualAge COBOL also features object-oriented extensions to COBOL, which enable you to create object-oriented programs in a language familiar to you.

What You Can Do Using VisualAge COBOL

VisualAge COBOL enables you to create stand-alone OS/2 COBOL applications—with or without graphical user interfaces (GUIs)—and to integrate your COBOL applications across a client/server environment.

Create OS/2 COBOL Applications

If you are creating a COBOL application with a GUI, VisualAge COBOL provides a GUI Designer from which you can create your interface, add the COBOL logic, and build the completed application.

You can also create traditional non-GUI COBOL applications. VisualAge COBOL provides the same set of visual, workstation application development tools to edit, compile, debug, and analyze your code.

Create Client/server Applications

VisualAge COBOL provides client/server support across a wide variety of platforms.

VisualAge COBOL enables you to create applications with OS/2 as your client and either OS/2, AIX, or MVS as the server. To do this, you need the appropriate compiler for the server platform.

Introducing VisualAge COBOL

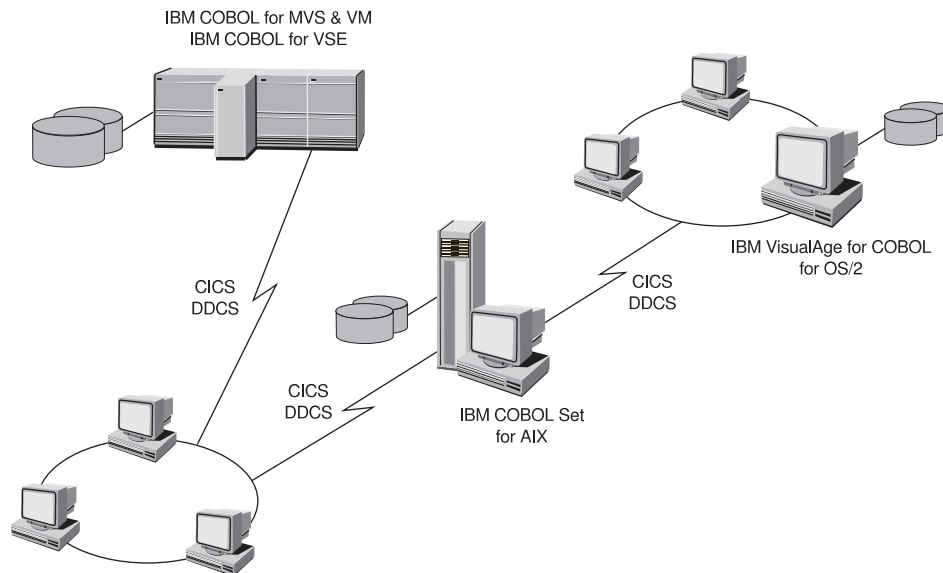


Figure 6. Client/server Computing Using IBM VisualAge for COBOL for OS/2. VisualAge COBOL provides client/server support across many platforms.

For data support, VisualAge COBOL gives you local and remote access to DB2 and VSAM. You can also access data managed by Btrieve**.

Create Object-Oriented Applications

VisualAge COBOL's object-oriented extensions offer greater opportunities for extending and reusing software. They enable you to develop new applications by defining data and the functions that operate on that data together in software modules called **classes**. This gives you the flexibility to change the structure of the data later without having any impact on the applications that access that data. These classes can then serve as the building blocks for reuse.

Your VisualAge COBOL Development Environment

VisualAge COBOL provides a complete development environment for creating many possible types of applications. WorkFrame integrates your development environment by organizing your workplace into projects—a logical grouping of files and their associated tools.

Edit, Compile, and Debug

VisualAge COBOL provides a set of visual tools to edit, compile, and debug your programs. When you set up your application project, these tools are available from the pop-up menus of your application files.

Introducing VisualAge COBOL

Create a GUI for Your Application

The visual GUI Designer enables you to build complex graphical user interfaces quickly and easily. You do not need in-depth GUI programming knowledge to create your application. With the GUI Designer, you select user interface controls and drop them onto your application window. These user interface controls enable you to build GUIs that conform to the IBM Common User Access (CUA) architecture guidelines. You then add the COBOL logic for your application using the GUI Designer and the language-sensitive COBOL Editor.

Create CICS ECI Calls and SQL Statements

The Transaction Assistant helps you generate a parameter list and the appropriate code for invoking CICS transactions. For example, your application can use a CICS transaction to invoke a server running under CICS on another system.

The Data Assistant enables you to visually map relational database information and generate syntactically-correct SQL statements in your COBOL applications.

Access Local and Remote Data

VisualAge COBOL provides you with access to your data in the following ways:

- VSAM data stored locally using the VSAM/2 record file system, and VSAM data stored remotely using VSAM APIs.
- Data managed by Btrieve using the file processing capabilities of COBOL.
- Local and remote DB2 data using DB2 for OS/2.
- Local and remote CICS data using CICS for OS/2.

Analyze Program Performance

You can tune and understand your programs by monitoring your program execution and generating a function-by-function trace of the run. This trace can be examined by utility programs that graphically display the execution trace.

Introducing VisualAge COBOL

Concepts for Developing an Application Using VisualAge COBOL

The VisualAge COBOL development paradigm centers around the concept of a *project*. A project is a container of your application files, such as COBOL source files, copy files, listings, object code, and executable files. These application files are known as *project parts*.

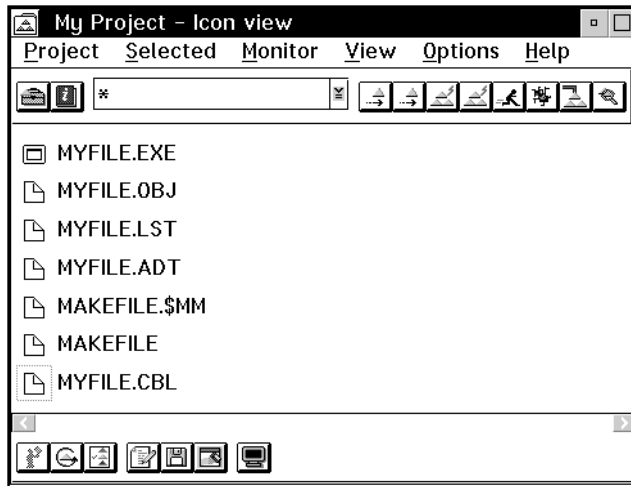


Figure 7. VisualAge COBOL Project. Projects contain your application files.

The project is set up to enable you to perform actions on those parts. The actions vary depending on the type of part. For example, an action appropriate for a COBOL source part would be edit. However, the edit action would not be appropriate for an executable file.

Introducing VisualAge COBOL

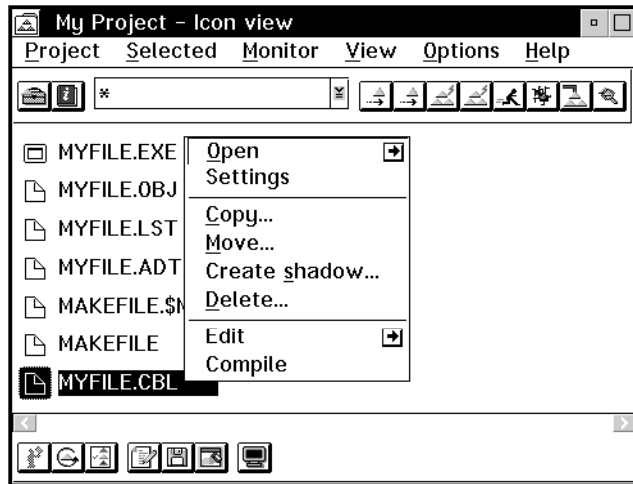


Figure 8. VisualAge COBOL Default Actions. VisualAge COBOL provides default, file-specific actions.

VisualAge COBOL supplies default projects that contain actions for the various types of parts you might have. You can modify existing actions and add new actions. For example, you may have a tool that searches through your COBOL source for COPY statements. You can add an action that would enable this tool to be invoked for all COBOL source files.

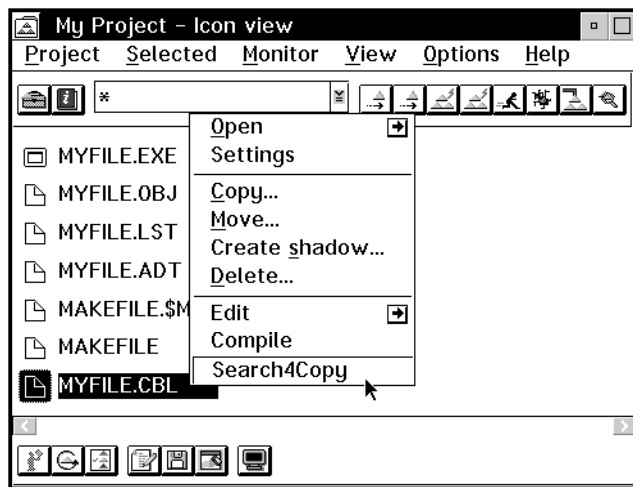


Figure 9. Customized VisualAge COBOL Actions. You can customize a COBOL project with your own tools.

Introducing VisualAge COBOL

Build Your First VisualAge COBOL Application

This chapter will guide you through building your first VisualAge COBOL application. The steps you follow here teach you the basic principles that you will use for further applications that you build. When you finish, you will have an application that displays a customized greeting.

Figure 10 shows you what the application's interface will look like when you have finished.

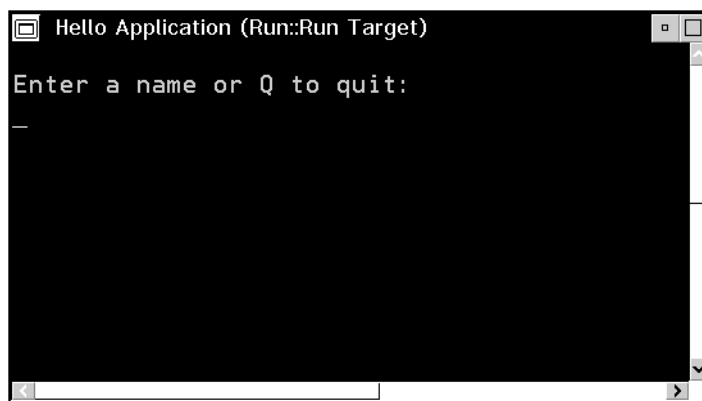


Figure 10. Hello Application. A COBOL application you can build using VisualAge COBOL.

Notice that this is an application that does not have a graphical user interface (GUI); you will build a GUI application in "Build Your First VisualAge COBOL GUI Application" on page 49.

The main steps you will follow are:

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

This chapter assumes that you are familiar with using the OS/2 interface. If you are not familiar with OS/2, either take the OS/2 Tutorial (normally located in the Information folder on the OS/2 desktop) or review Appendix A, "Using OS/2" on page 131.

Creating the Project

Your first step in developing an application with VisualAge COBOL is to set up a project. A project contains all of the components you need to build a target, for example, the files you need to create an application. The Hello Application project you

Creating the Project

create in this chapter contains a COBOL source file (a component or project part) from which you build the running COBOL program (a target).

To set up a new project:

1. Double-click on the **VisualAge COBOL** icon. The VisualAge COBOL - Icon View window opens.
2. Double-click on the **Create New Project** icon. The IBM VisualAge COBOL - Create New Project window opens, as shown in Figure 11.

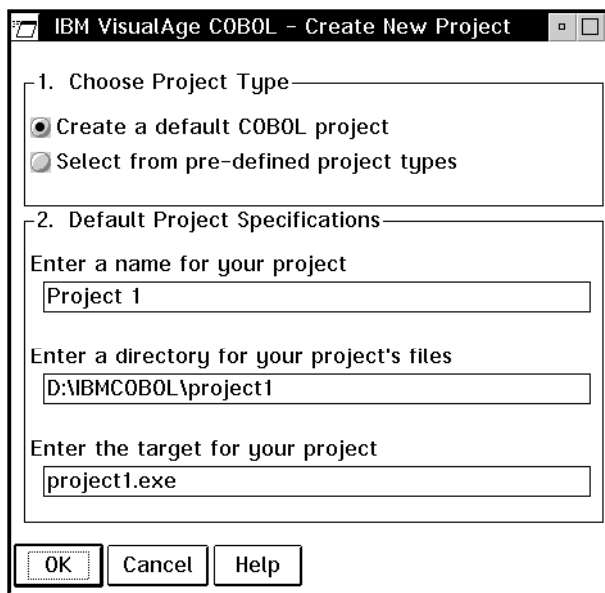


Figure 11. Create New Project window. The starting point for creating COBOL projects.

3. In the **Choose Project Type** group, ensure that the **Create a default COBOL project** radio button is selected.
4. In the **Default Project Specifications** group, specify your project's name and the location where the project's files are stored. In the entry field titled **Enter a name for your project**, enter the name Hello Application.
5. In the entry field titled **Enter a directory for your project's files**, enter the directory where you are planning to put your source files. If you click in the entry field, the name is automatically updated to \HelloApp. The name is also updated to HelloApp.exe in the **Enter the target for your project** entry field.
6. Click on the **OK** push button to create the project. Since the directory for the Hello Application (\HelloApp) has not yet been created, a message window appears. Click on the **OK** push button. The directory is created and the **Hello Application - Icon view** window opens.

Creating the Application

Now that you have created your project, you can create the files you need for the application.

Creating the Application

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

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

1. From the Hello Application - Icon view window, select the **Project** menu-bar choice, then select the arrow button next to the **Create** choice.
2. A cascaded menu appears. From the cascaded menu, select **Create New Text File with the COBOL Editor**. The New window appears, in which you specify options for your new file.
3. In the New 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. This sets the COBOL language-sensitive editing features on.
4. Click on the **New** pushbutton. The COBOL Editor displays the window titled Editor - Untitled Document 1 and recognizes the file as a COBOL file.

You can tell that the language-sensitive editing features are on by checking to see that the format line, the line just below the menu bar, displays *, A, and B, as shown in Figure 12. If you cannot see the format line, you can display it by selecting **View**, then selecting **Format line**.

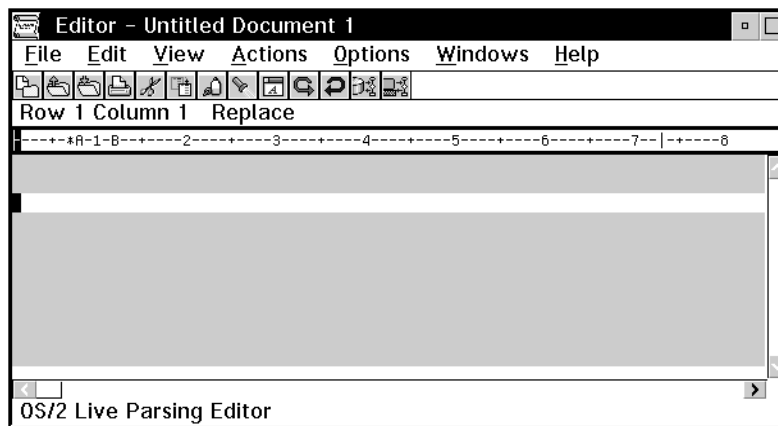


Figure 12. Editor window. The COBOL Editor shown with language-sensitive editing enabled.

5. Enter the source code shown in Figure 13 on page 46.

Creating the Application

```
*****
IDENTIFICATION DIVISION.
*****

PROGRAM-ID.    HELLOAPP.
AUTHOR.        Tester.

*****
ENVIRONMENT DIVISION.
*****
CONFIGURATION SECTION.
*SOURCE-COMPUTER.
*OBJECT-COMPUTER.

*****
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.
      88 LOOP-NOT-DONE         VALUE 0.

*****
PROCEDURE DIVISION.
*****

0000-MAINLINE.

    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.

0000-EXIT.
GOBACK.
```

Figure 13. COBOL Source Code for the Hello Application

- When you have finished entering the source code, save the file. To save the file, select the **File** menu-bar choice, then select **Save as**. Ensure that the **Directory** list box shows HELLOAPP as the selected directory. Enter helloapp.cb1 in the **Save as filename** field and click on the **Save As** push button. Then close the COBOL Editor by double-clicking on the system-menu symbol in the upper-left corner of the window.

Building the Application

7. If the file doesn't appear in the Hello Application - Icon view window, make the window active by clicking on its title bar, then press **F5** (refresh).

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 just have a single COBOL source object, which you build into a running COBOL program.

1. Select the **Project** menu-bar choice, then select **Build**. From the cascaded menu, select **Build normal**.
2. The target, an executable file titled `helloapp.exe`, is created from the COBOL source file. As the build runs, you can see its progress in the Monitor. The Monitor appears at the bottom of the Hello Application - Icon view window.
3. When the build is complete, the Monitor displays the return code and new files appear in the Hello Application - Icon view window, as shown in Figure 14.

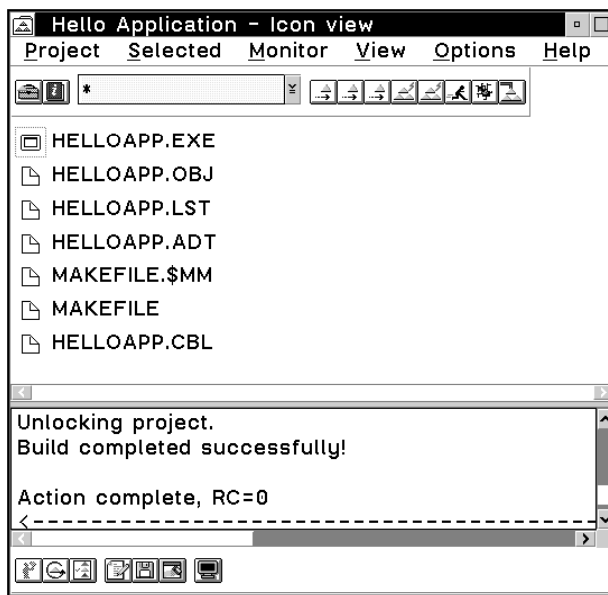


Figure 14. Hello Application - Icon view window with Monitor. Monitor shows progress of the build and the final return code.

A return code of 0 means that the target was built without errors.

If you do not get a return code of 0, the Monitor window displays error messages. Scroll back up to the error message lines that have the drive, path names, and source file name (`helloapp.cbl`) as well as the error message text. Some linker

Running the Application

error messages also contain drive, path, and file names. These are the compiler error messages. Double-click on a compiler error message line.

Note: You can only double-click on the compiler error messages to fix errors in the source program.

The COBOL Editor appears, showing the line in the `helloapp.cbl` source file where the error occurred. Correct the error and save the file. To build it again, select the **Project** menu-bar choice, then select **Build**. You will see the results of the second build in the Monitor window.

You are now ready to run the Hello Application.

Running the Application

You can run your application from the Hello Application - Icon view window. To run the Hello Application, select the **Project** menu-bar choice, then select **Run**. An OS/2 window appears and runs your program.

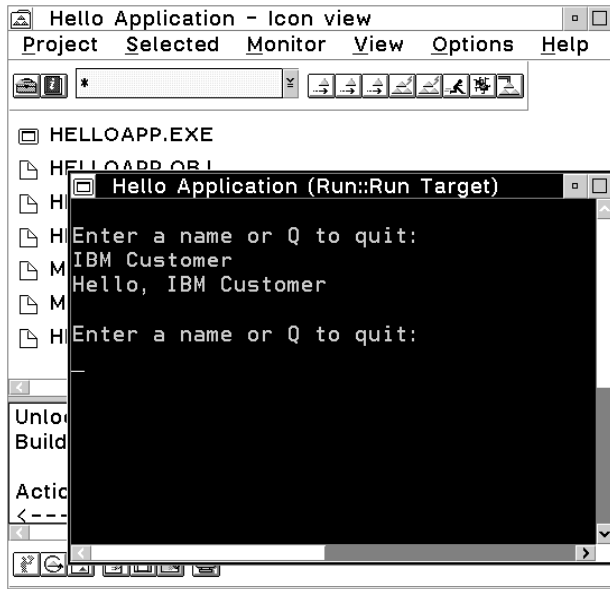


Figure 15. Hello Application. The Hello Application in action....

Build Your First VisualAge COBOL GUI Application

This chapter will guide you through building your first VisualAge COBOL application that has a graphical user interface (GUI).

When you finish, you will have an application similar to the one you developed in “Build Your First VisualAge COBOL Application” on page 43, but the application will have a GUI. Although there are some differences when you create an application that has a GUI, many of the tools are as those you used to build the character-based Hello application. Figure 16 shows you what the application's GUI will look like when you have finished.

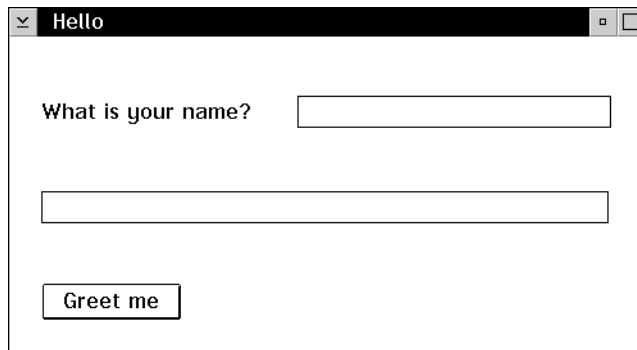


Figure 16. Hello GUI Application. A COBOL application you can build using VisualAge COBOL.

The main steps you will follow are:

- Creating the GUI project
- Creating the graphical user interface
- Creating the application logic
- Building the application
- Running the application

This chapter assumes that you are familiar with using the OS/2 interface. If you are not familiar with OS/2, either take the OS/2 Tutorial (normally located in the Information folder on the OS/2 desktop) or review Appendix A, “Using OS/2” on page 131.

Creating the GUI Project

Your first step in developing a GUI application with VisualAge COBOL is to set up a GUI project. Like other projects, the GUI project contains the components you need to create a specific target; for example, the files you need to create an application.

To set up a new GUI project:

1. Double-click on the **VisualAge COBOL** icon. The VisualAge COBOL - Icon View window opens.
2. Double-click on the **Create New Project** icon. The IBM VisualAge COBOL - Create New Project window opens.
3. In the **Choose Project Type** group, select the **Select from predefined project types** radio button and click on the **OK** push button.
4. The COBOL Project Smarts - Catalog View window opens, as shown in Figure 17.

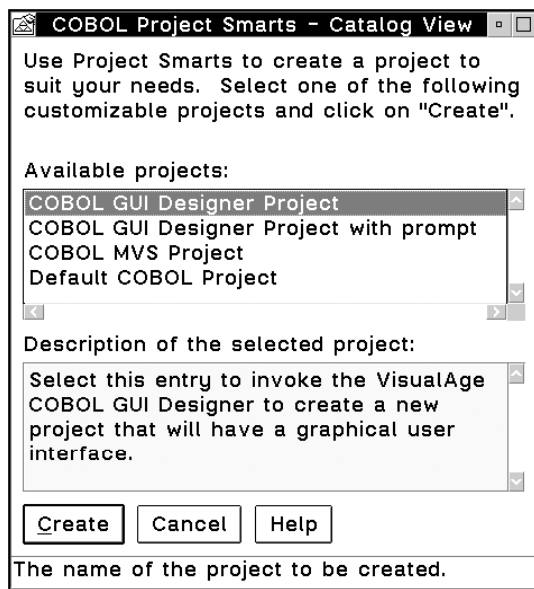


Figure 17. COBOL Project Smarts - Catalog View. Create COBOL projects from the COBOL Project Smarts.

From the **Available Projects** list box, select **COBOL GUI Designer Project** and click on the **Create** push button.

5. The COBOL GUI Designer opens. This may take a few moments.

Creating the Graphical User Interface

Now that your project is set up, you can create the GUI for the application. You will use the COBOL GUI Designer to create the GUI for the Hello GUI Application.

The COBOL GUI Designer enables you to build GUIs by dragging and dropping the GUI parts, such as a check box, onto a window part. It also enables you to create the logic that runs the GUI application.

Since you chose to create a GUI Project during the project set-up, the COBOL GUI Designer - Untitled, Parts Palette, and Window with canvas windows should appear on your desktop.

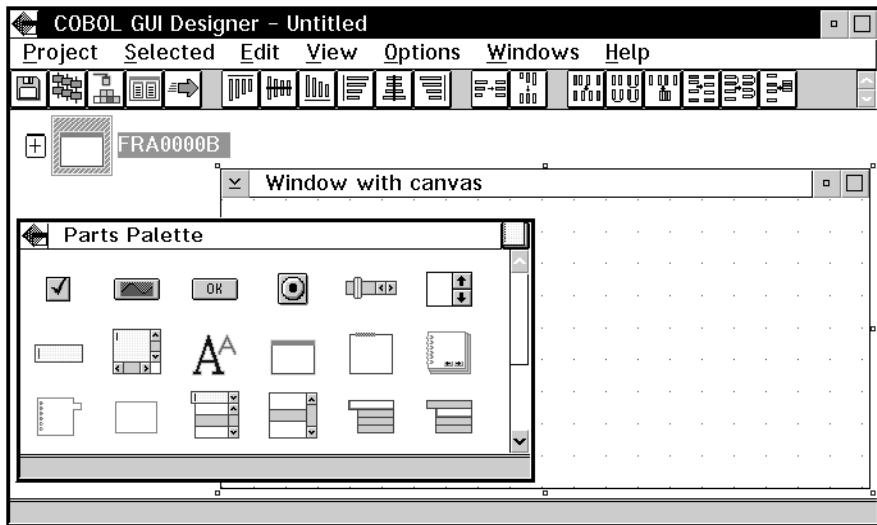


Figure 18. COBOL GUI Designer. GUI Designer window with Parts Palette and Window with canvas.

The COBOL GUI Designer - Untitled window is the main window. When you initially open this window, it assumes you want to create a new GUI. The COBOL GUI Designer creates a new window part for you, titled **Window with canvas**.

The Parts Palette contains the parts you use to construct your GUI. You use your mouse pointer to move parts onto the Window with canvas.

Notice that when you move the mouse pointer over an object in the Parts Palette, the name of the object appears in the *information area* at the bottom of the Parts Palette window. Make sure that the entire Parts Palette window is visible so you can see the information area at the bottom of the window. This will help you identify the parts.

If you are not familiar with GUI parts, the Parts Catalog might be helpful to you as you learn to use the COBOL GUI Designer. The Parts Catalog is a notebook with the GUI parts labeled and grouped into categories, as shown in Figure 19 on page 52.

Adding the GUI Parts

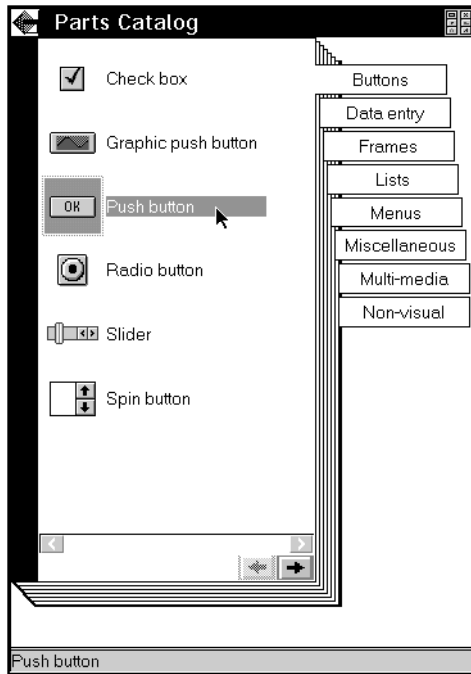


Figure 19. COBOL GUI Designer - Parts Catalog notebook. A helpful way to access GUI parts.

To change the Parts Palette window to display the Parts Catalog, click on the small icon that resembles a notebook in the upper right corner of the Parts Palette window. This icon acts as a toggle: the window changes to display the **Parts Catalog** notebook and the icon changes to a palette. When you click on the icon in the upper right corner again, the notebook changes back to the Parts Palette window.

Note: Although you can easily switch from the Parts Palette to the Parts Catalog and back again, these windows are not identical. Parts may be added to the Parts Catalog that are not automatically reflected in the Parts Palette.

Adding the GUI Parts

Using the Parts Palette or the Parts Catalog, drag and drop¹ the following parts onto the Window with canvas, as shown in Figure 20 on page 53.

As you place the parts on the Window with canvas, make sure that you align your parts with the *bottom* margin of the window. This ensures that if you change the size of the window, your parts will stay in view.

Customizing the GUI

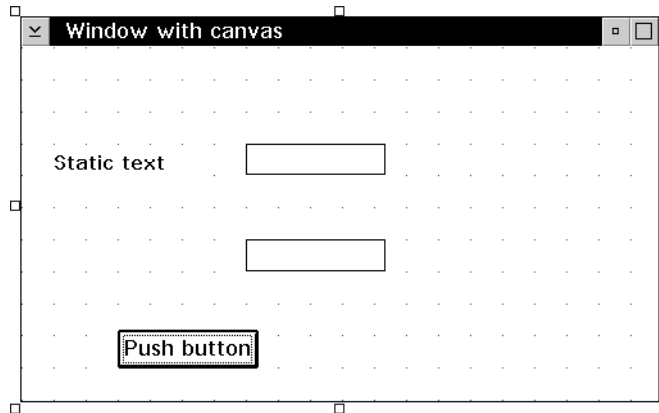


Figure 20. COBOL GUI Designer with GUI Parts. Window with canvas shown with the GUI parts for the Hello GUI.

1. Drag and drop one **Static Text** field onto the window.
2. Drag and drop two **Entry** fields onto the window. The first should be between the static text field and the right border of the window. The second should be beneath the first entry field.
3. Drag and drop one **Push button** between the second entry field and the bottom of the window.

Customizing the GUI

When you have parts on the window, you can change the parts to customize them for your application.

For the Hello GUI Application, you need to change some of the part attributes. You can change these attributes by using the settings notebook.

The *settings notebook* is used to set the initial run-time attributes for parts on your interface, such as color and font. You can also use the settings notebook for a part to change the part's name, which your program uses to query and change the part's appearance while the program runs.

First, change the attributes for the first Entry field part:

1. Double click on the first **Entry** field part to open the **Entry Field Part Settings** notebook.
2. Move the cursor to the **Part name** entry field and change the contents to NAME.

The part name is the internal name by which your part will be known. This name will also appear in your COBOL program and in the VisualAge COBOL tools that

¹ To drag and drop an object, move the mouse pointer over the object you want to move or copy and press mouse button 2. Move the object over the place you want it, then release the mouse button.

Customizing the GUI

assist you in creating the COBOL code. You don't have to change the part name. By giving it a meaningful name, as you would with a variable in a program, the part will be easier to identify when you are creating the Hello GUI Application logic.

When you create your own GUI programs, ensure that you have the part names you want corresponding to the correct GUI part.

3. Select the **Data** tab and set the length of the entry field to **30**.

When you've completed the steps above, the settings notebook should resemble Figure 21.

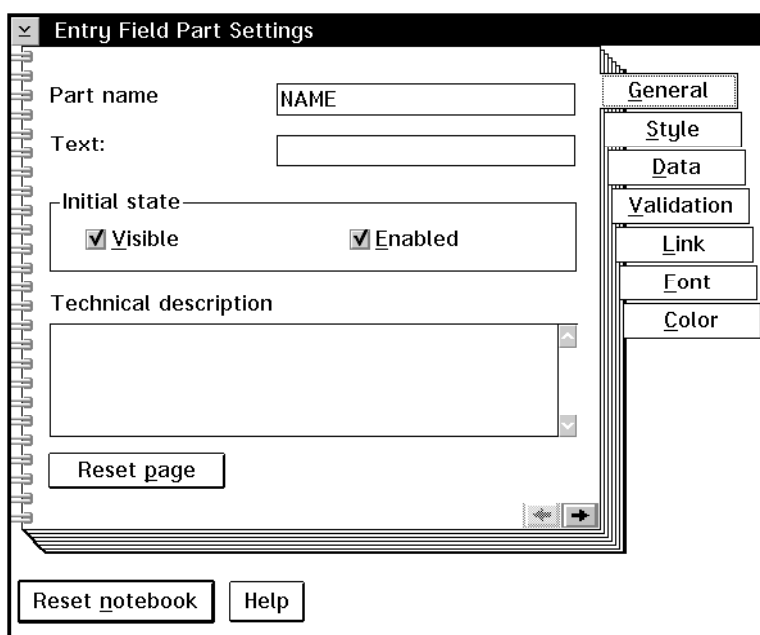


Figure 21. Settings Notebook for the Entry Field Part. **Entry Field Part Settings** with new part name.

4. Double-click on the system-menu symbol at the upper left corner of the settings notebook window. Your changes are saved.

Next, change the attributes for the rest of your parts:

- For the second Entry field part, change the part name to GREETING. Select the **Style** tab and click on the **Read-only** check box.
- For the push button, change the part name to GREETME and the label to Greet me.
- For the static text part, change the text to What is your name?.
- For the Window with canvas part, double-click on the **title bar** of the Window with canvas. Change the part name to HELLOGUI and the title to Hello.

Saving the GUI Project

Moving and Sizing GUI Parts

You may need to move or change the size of the parts to fully view the labels or see the contents of entry fields.

To move a part:

1. Select the part by clicking on it with mouse button 1.
2. Move the mouse pointer over the part.
3. Click and hold mouse button 2.
4. Move the mouse to the place you want the part.
5. Release the mouse button.

To size a part:

1. Select the part by clicking on it with mouse button 1. Small boxes, or “handles,” appear around the part.
2. Move the mouse pointer over a handle, so that the mouse pointer changes to a double-headed arrow.
3. Press and hold mouse button 2.
4. Move the mouse until the part is the size you want.
5. Release the mouse button.

When you have finished, the GUI window should resemble the window in Figure 22.

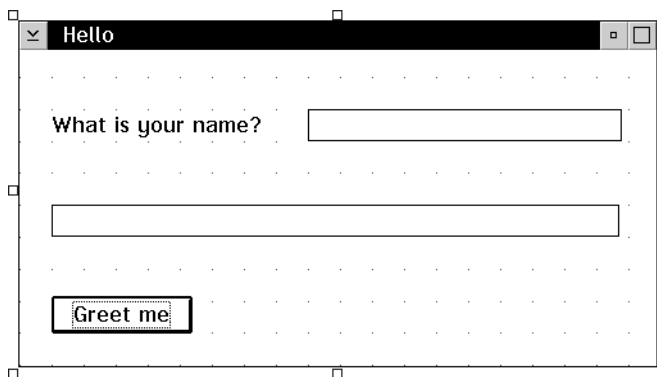


Figure 22. The completed Hello GUI window design

Saving the GUI Project

To save the GUI project, select **Project**, then select **Save**, as shown in Figure 23 on page 56.

Saving the GUI Project

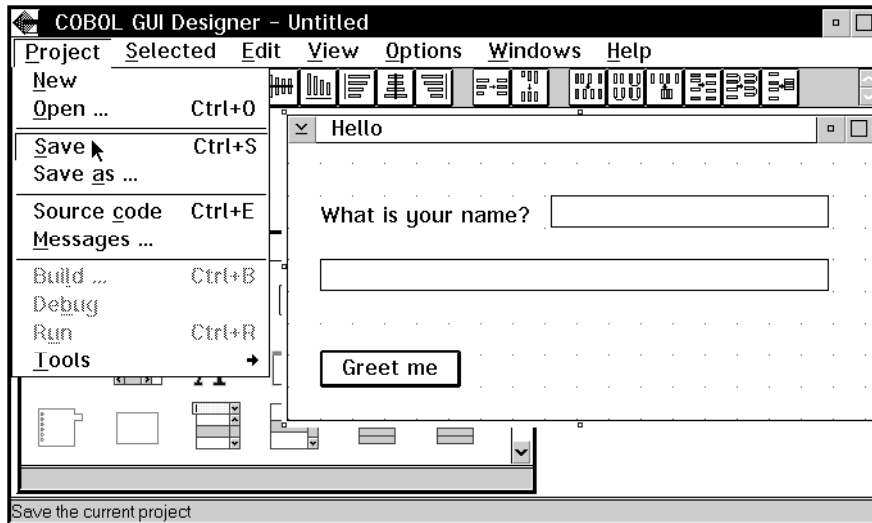


Figure 23. Saving the COBOL GUI Project

The COBOL GUI Designer - Save as Application window appears. When you save a new application, COBOL GUI Designer - Save as Application window enables you to select the name and location of your project.

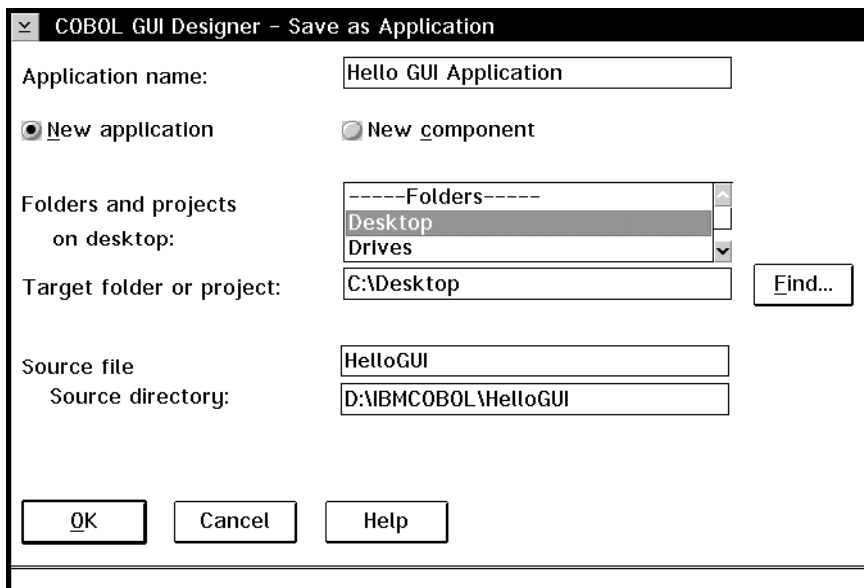


Figure 24. COBOL GUI Designer - Save as Application window

For the Hello GUI Application, perform the following steps:

1. Type Hello GUI Application in the **Application Name** field.

Event-Driven Programming

2. Ensure that the **New application** radio button is selected (it is the default).
3. For **Folders and projects on desktop**, select the desktop item (**Desktop** by default).
4. In the **Source file** field, the file name is **HelloGUI**. This defaults to the first 8 characters of the Application name—in this example, **Hello GUI Application**. You can change the default as desired.
5. In the **Source directory** field, the default directory displayed is a modification of the current directory. For example, if you are in the directory C:\IBMCOBOL and your source file name is HelloGUI, the default source directory is C:\IBMCOBOL\HelloGUI. You can change the default as desired.
6. Click on **OK**.

The Hello GUI Application project is created. The COBOL GUI Designer window title changes from **Untitled** to your new application name. For the Hello GUI Application, it changes to **Hello GUI Application**. The project files are saved in the path you specified in the **Source directory** field. The next time you save your work, select the **Project** menu-bar choice, then select **Save**.

Note: When you create your own GUI projects, you can save the GUI project at any point before you build the GUI project.

Creating the Application Logic

Your next step for developing the Hello GUI Application involves coding the logic behind the parts in your GUI. The COBOL program behind your GUI follows the *event-driven programming* paradigm.

Event-Driven Programming

If you have developed COBOL applications on a mainframe system, you are familiar with *procedural programming* techniques. If you plan to develop GUI applications with the COBOL GUI Designer, you need an understanding of *event-driven programming* techniques.

There is one main difference between these two programming techniques: an application developed using procedural programming has one entry and one exit point, but an application developed using event-driven programming has many entry and exit points. These entry and exit points correspond to the different events that cause the program to take action; for example, the user interacting with the user interface.

When you write a COBOL program using procedural programming, the program has one point of entry and exit. The program follows each step in the program logic sequentially until it reaches the end of the logic.

Event-Driven Programming

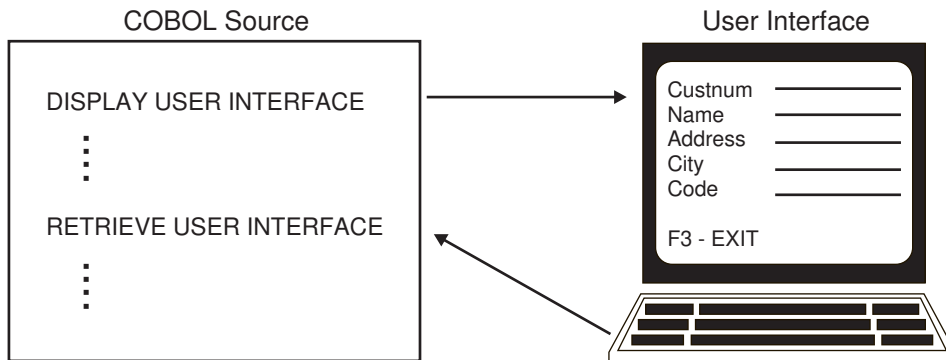


Figure 25. Procedural Programming

An *event-driven program* is a program that runs segments of logic in response to events. It has entry and exit points that correspond to the 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 the user clicks the **Greet me** push button. Once a selected event occurs, only the logic for that event is performed, then the application waits for the next event.

When you write a COBOL program using the COBOL GUI Designer, you use *event-driven programming* techniques. The logic flow is determined by the events that you have chosen for the parts in your graphical user interface. With the help of the COBOL GUI Designer, you can code COBOL logic to respond to these events. Event logic is identified in your COBOL program by an ENTRY statement. The statements between each set of ENTRY and GOBACK statements (including the ENTRY and GOBACK) are executed when the corresponding event for the logic has been signalled. Each graphical user interface part responds to a number of events.

Event-Driven Model

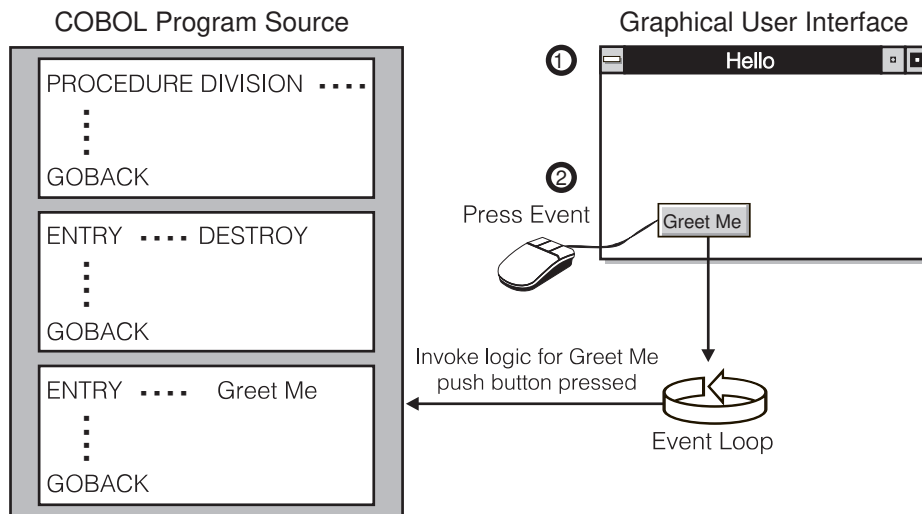


Figure 26. Event-Driven Programming

For example, in Figure 26:

1. A window is displayed.
2. When the end-user triggers an event in the window, the appropriate logic is performed. In this case, the user chooses the **Greet me** push button and the corresponding logic is performed. The program then waits for the next event to occur.

Creating the Event Logic

The COBOL GUI Designer helps you code the event logic. Each part you place on the user interface has a pop-up menu choice titled **Events**, from which you select the event to which your logic will respond.

Once you've selected the event, the COBOL GUI Editor window appears with the COBOL program for your GUI application and the cursor is positioned below the ENTRY statement for the selected event. This COBOL program initially contains the basic COBOL program constructs, such as DATA DIVISION. Several COPY statements are included as required for a GUI application. Other than the PROCEDURE DIVISION/GOBACK statement pair, between which you can code initialization logic to be executed before your GUI is displayed, only one event is already coded. That event is the window destroy event and is executed when your GUI application's window is closed.

You now add the code you want to run for the event. When you need to access the screen data and attributes in your code, you can use the **Edit** menu-bar choice and select **Insert code**→**GUI** to get assistance with creating the logic. Figure 27 on page 60 summarizes the steps you take to create event logic.

Creating the Event Logic

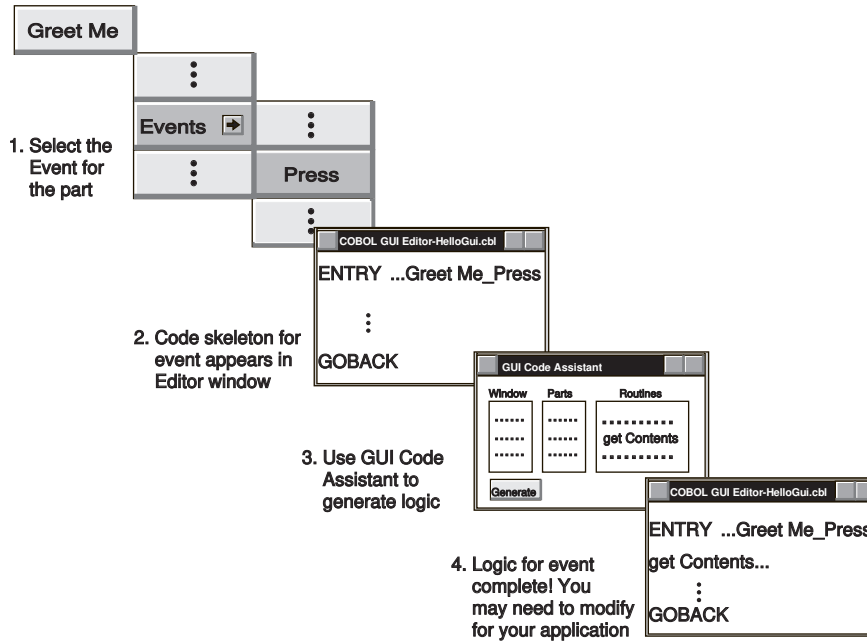


Figure 27. Steps for Coding the Event Logic. The COBOL GUI Designer helps you code your event logic.

The following steps show you how to create logic for the Hello GUI Application. When a user types a name—for example, IBM Customer—in the **What is your name?** entry field and presses the **Greet me** push button, the message “Hello, IBM Customer” appears in the second entry field.

You will create the logic to do this. First, you will indicate what the event is that your logic will respond to—for the Hello GUI Application, the logic responds to the **Greet me** push button being pressed.

Note: Before you begin, make sure that the part names correspond to the correct parts of your GUI; for example, that the Window with canvas has a part name of HELLOGUI. If you change the part name of a window or control after you have started to create the event code, you must change the part names in the affected program logic. These changes are not made automatically when you change the part name.

1. Move the mouse pointer to the **Greet me** push button on the Hello window and press mouse button 2. A pop-up menu appears.
2. Select **Events**, then select **PRESS** from the cascaded menu. You will be specifying the logic that is to be run when the user clicks on (or presses) the **Greet me** push button.

Creating the Event Logic

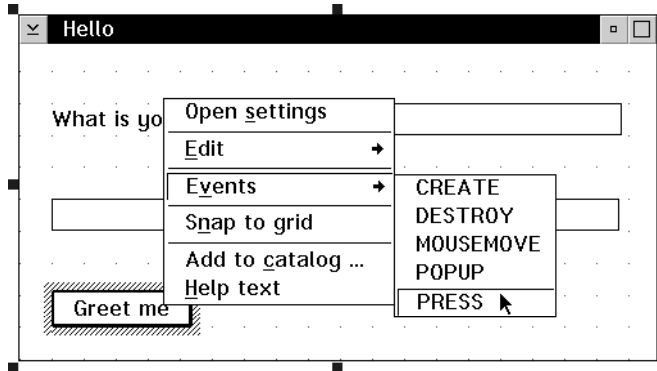


Figure 28. Selecting the PRESS Event for the Greet me Pushbutton

3. The COBOL GUI Editor window opens at the COBOL ENTRY statement for the **Greet me** push button press event.

Notice the format of the ENTRY statement, ENTRY "HELLOGUI_GREETME_PRESS" USING VDE-HELLOGUI. The entry point name includes the window name, the part name, and the event selected; these names are separated by underscores.

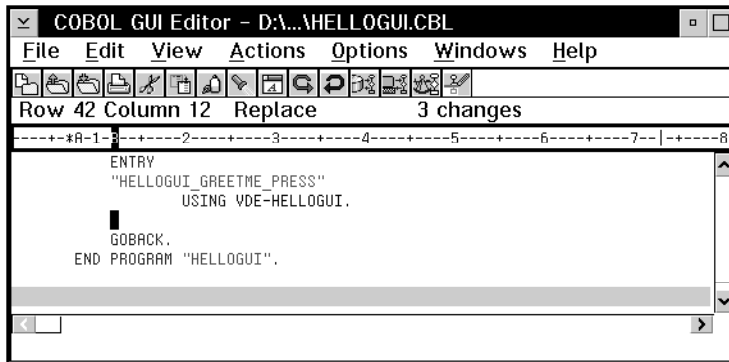


Figure 29. Editor window with COBOL ENTRY statement for the PRESS Event

At this point, you need to specify what happens when the **Greet me** push button is pressed. The application logic must first retrieve the contents of the **NAME** entry field, add the greeting text, and move the greeting to the second entry field where it will be displayed for the user. VisualAge COBOL provides assistance to help you generate the COBOL code to work with the GUI part.

1. Click on the title bar of the COBOL GUI Editor window. The cursor in the window should appear on the line below the ENTRY statement (after USING VDE-HELLOGUI). If the cursor isn't positioned there, move it there now.

Select the **Edit** menu-bar choice on the editor window, then select **Insert code**. From the cascaded menu, select **GUI**. The GUI Code Assistant window opens, as shown in Figure 30 on page 62.

Creating the Event Logic

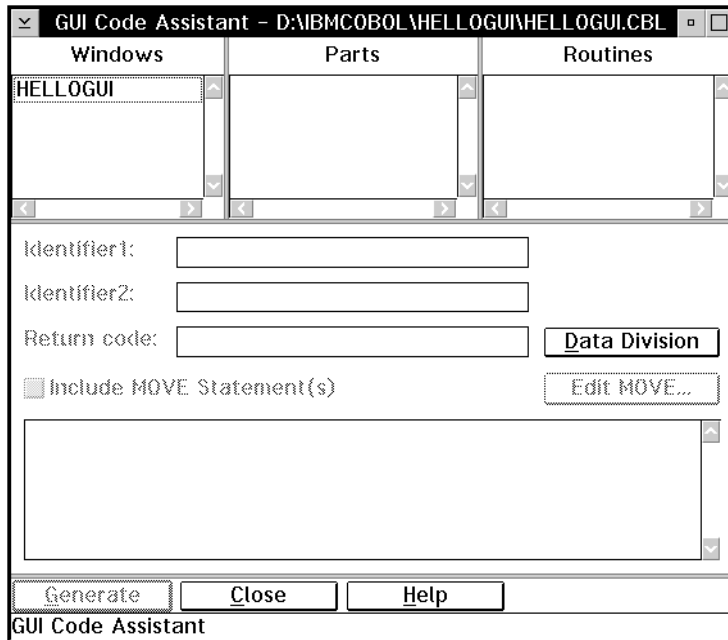


Figure 30. GUI Code Assistant

2. Click on **HELLOGUI** in the **Window** list box to select the window that contains the **NAME** entry field, the part from which you want to receive the contents.
3. Click on **NAME** in the **Parts** list box.
4. Click on **getContents** in the **Routines** list box to get the value of the **NAME** entry field. You may need to scroll the **Routines** list box to find **getContents**.
5. Notice the CALL statement in the multiple-line entry field just above the push buttons at the bottom of the window. This is the statement that will be generated. Click on the **Generate** push button to complete.

A CALL statement is generated that gets the value of the **NAME** entry field, as shown in Figure 31 on page 63. This CALL statement issues a call to the routine **getContents**, using parameters generated by the GUI Code Assistant. These parameters are based on the windows, parts, and routines you select.

Note: When you generate code using the GUI Code Assistant window, the GUI Code Assistant uses default variables. These variables are defined in a copy file (VACCESS.CPY) that is automatically included in this program. Many of the variables in this copy file are variable-length tables.

Creating the Event Logic

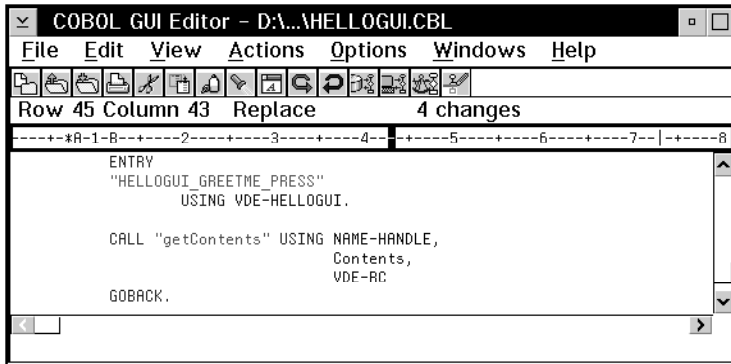


Figure 31. COBOL Logic Generated Using the GUI Code Assistant

6. To close the GUI Code Assistant window, click on the **Close** push button.

The next action of this event is to create the greeting string using the value the user enters. You need to add code to the event logic.

1. Move the cursor in the COBOL GUI Editor window to the line after the logic created to get the contents of the **NAME** entry field (CALL "getContents" USING NAME=HANDLE, Contents, VDE=RC).
2. Enter the following code. The variables in the code below are defined in the VACCESS.CPY file. To add extra blank lines, press the Enter key.

```
ADD 7 Contents-Length GIVING NewText-Length  
MOVE "Hello, " TO NewText-String(1:7)  
MOVE Contents-String(1:Contents-Length)  
  TO NewText-String(8:Contents-Length)
```

You now have the logic for adding the string "Hello, " to the name the user enters into the first entry field.

The final action of this event is to move the text to the entry field in which the greeting is to be displayed. You can use the GUI Code Assistant window again to generate the code.

1. In the editor window, place the cursor on a blank line between the code you just entered and the GOBACK statement.
2. Select **Edit** from the COBOL GUI Editor menu bar. Select **Insert code**, then select **GUI**.
3. When the GUI Code Assistant window appears, select the **GREETING** part and the **setContents** routine in the GUI Code Assistant window.
4. In the **Identifier 1** entry field, change the default, Contents, to the variable to which you have moved the text to be displayed, NewText.
5. Select the **Include MOVE Statement(s)** check box to uncheck it.
6. Select **Generate** in the GUI Code Assistant window. The code you generated appears in the editor window.

Creating the Event Logic

7. To close the GUI Code Assistant window, click on the **Close** push button.
8. To save your changes, select the **File** menu-bar choice then select **Save**. Close the editor window by double-clicking on the system-menu symbol in the upper left corner of the window.

You just created the logic for the **Greet me** push button.

The logic to close the GUI application has already been coded for you. To view this event logic:

1. Click mouse button 2 on the Hello title bar.
2. Select **Events**.

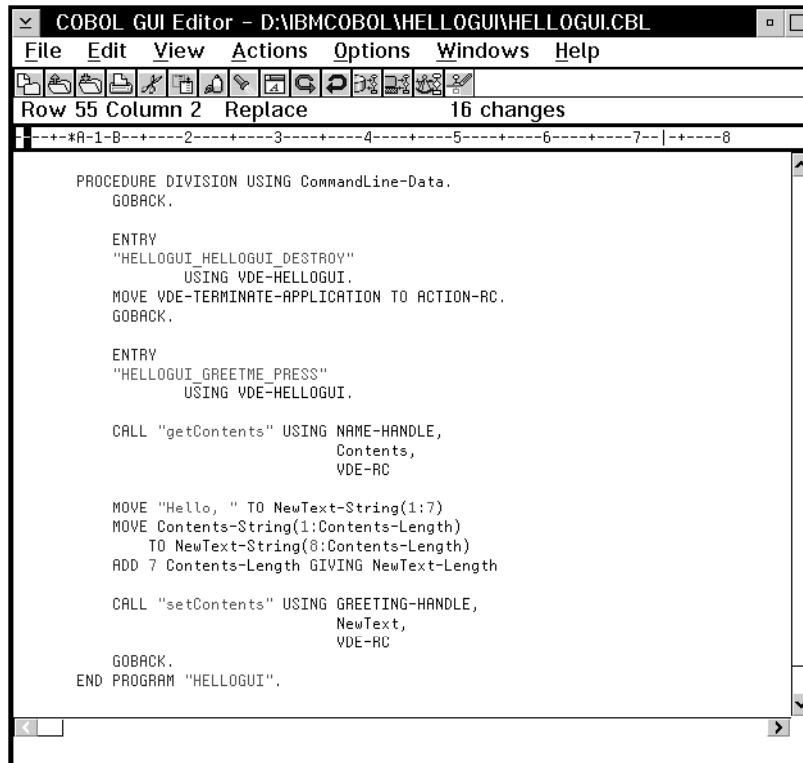
Notice the check mark on the **DESTROY** menu choice on the **Events** cascaded menu. This means that the event logic is already created. Clicking on the menu choice takes you to the existing event logic.

3. Select **DESTROY**.

The COBOL GUI Editor window appears. The cursor is placed in the source code at the DESTROY event. The event code contains the statement `MOVE VDE-TERMINATE-APPLICATION TO VDE-RC`. This logic causes the application to end when the user closes the GUI window.

Figure 32 on page 65 shows the COBOL GUI Editor window with the event code you've added for the Hello GUI application.

Building the Application



The screenshot shows a window titled "COBOL GUI Editor - D:\IBMCOBOL\HELLOGUI\HELLOGUI.CBL". The menu bar includes File, Edit, View, Actions, Options, Windows, and Help. The status bar indicates "Row 55 Column 2 Replace 16 changes". The main text area contains the following COBOL code:

```
PROCEDURE DIVISION USING CommandLine-Data.
  GOBACK.

  ENTRY
    "HELLOGUI_HELLOGUI_DESTROY"
    USING VDE-HELLOGUI.
  MOVE VDE-TERMINATE-APPLICATION TO ACTION-RC.
  GOBACK.

  ENTRY
    "HELLOGUI_GREETME_PRESS"
    USING VDE-HELLOGUI.

  CALL "getContents" USING NAME-HANDLE,
    Contents,
    VDE-RC

  MOVE "Hello, " TO NewText-String(1:7)
  MOVE Contents-String(1:Contents-Length)
    TO NewText-String(8:Contents-Length)
  ADD 7 Contents-Length GIVING NewText-Length

  CALL "setContents" USING GREETING-HANDLE,
    NewText,
    VDE-RC

  GOBACK.
END PROGRAM "HELLOGUI".
```

Figure 32. Event Logic for the Hello GUI Application

Close the COBOL GUI Editor window. You are now ready to build the Hello GUI Application.

Building the Application

When you build a VisualAge COBOL GUI application, the source files are compiled and linked to create a running application. To build your application:

1. From the COBOL GUI Designer - Hello GUI Application window, select the **Project** menu-bar choice.
2. Select **Build**.
3. When the build starts, the Project window appears.

If you did not save the project before the build, the COBOL GUI Designer - Save Project window appears. Select **Save** to save your project.

Within the Project window is the Monitor, which displays the output of the build. The files that you created for the application are compiled and linked to create the specified target file.

Building the Application

- When the build completes, the return code is displayed, as shown in Figure 33 on page 66. A return code of zero indicates that your application was built without errors and is ready to run.

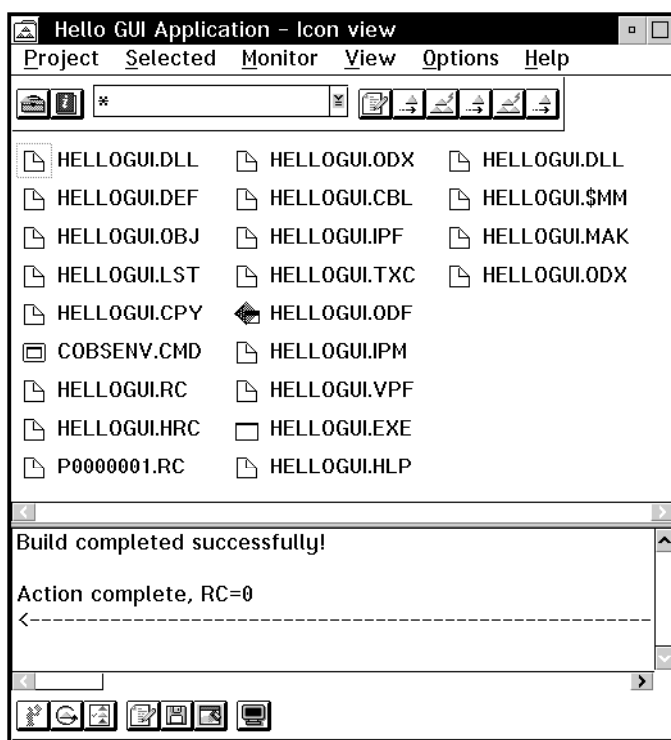


Figure 33. Building the Hello GUI Application. The Hello GUI Project window with the Monitor Area at bottom.

If you do not get a return code of 0, the Monitor window displays error messages. Scroll back up to the error message lines that have the drive, path names, and source file name (hellogui.cbl) Some linker error messages also contain drive, path, and file names. as well as the error message text. These are the compiler error messages. Double-click on a compiler error message line.

Note: You can only double-click on the compiler error messages to fix errors in the source program.

The COBOL GUI Editor appears, showing the line in the source file where the error occurred. Correct the error, save the file, and close the editor. To build it again, select **Project** in the project window, then select **Build**. You will see the results of the second build in the Monitor window.

Running the Application

Running the Application

To run the Hello GUI Application, select the **Project** menu-bar choice, then select **Run**. The Hello window appears, and you can try out your GUI application.

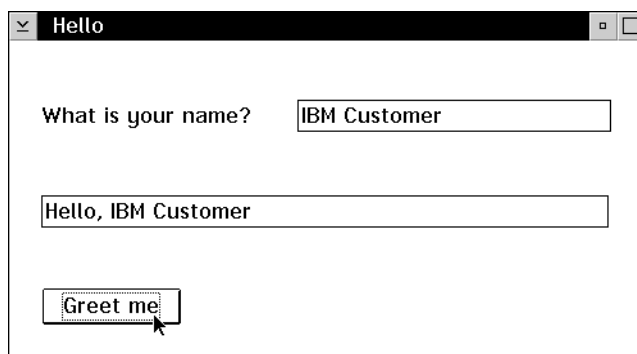


Figure 34. Running the Hello GUI Application

To close the Hello GUI Application, double-click in the upper left corner of the window.

Running the Application

Tools in VisualAge COBOL

VisualAge COBOL consists of several application development tools, integrated with WorkFrame. This section provides an overview of these tools.

WorkFrame

WorkFrame for OS/2 integrates your tools and files for the VisualAge COBOL application development environment.

It enables you to work with your files directly rather than accessing them through a particular tool. 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 WorkFrame to provide context-sensitive actions for your files.

WorkFrame enables you to organize your code by grouping related files into projects. A *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). It consists of *project parts*, the data objects that make up the projects, and a *Tools setup*, the actions, environment variables, and types that are available to a project.

Your WorkFrame project is fully customizable. You can include objects of many different types in a project. You can choose the tools available for actions on these objects, such as editors, compilers, and debuggers. You can customize your project not only with your own OS/2 tools, but DOS and Windows tools as well.

VisualAge COBOL provides pre-configured projects as a part of the product. When you create your own projects, you can *inherit* the complete set of actions, types, and environment variable settings from these default projects. This means that you get an environment with the tools and actions already set up for you. The **Create New Project** icon in the VisualAge COBOL folder enables you to easily start a project that inherits from a default COBOL project.

COBOL Editor

The COBOL Editor provides language-sensitive editing for your files. For example, different COBOL constructs, such as comments, are shown in different colors.

Aside from standard editing functions, you can issue coding assistance commands. You can create GUI code, generate an SQL statement, or generate a call to invoke a CICS program.

GUI Code Assistant

Generates code to access the data and attributes of your graphical user interface parts.

VisualAge COBOL Tools

Data Assistant

Generates SQL statements in your COBOL source.

Note: Before you can use the Data Assistant to create SQL statements, you need to map the appropriate data structure to a relational data base. You can do this visually; see the Data Assistant online help for more information.

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.

Note that the COBOL Editor displays certain toolbar icons and menu choices for the coding assistants only when you need them. Otherwise, the toolbar icons and menu choices are not visible on the menu. The toolbar 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). The COBOL GUI Editor, a specialized version of the COBOL Editor that contains the GUI Code Assistant, is not shown unless you start it from the COBOL GUI Designer.

COBOL GUI Designer

The COBOL GUI Designer is a tool that enables you to create COBOL GUI applications. The COBOL GUI Designer not only helps you quickly create a GUI application without having to write Presentation Manager code; it helps you create your whole COBOL program.

The GUI Designer is based on a “construction from parts” paradigm. It consists of a “parts palette” with the different GUI controls (such as windows, list boxes, entry fields). You create the GUI by dragging and dropping these parts into the GUI Designer window. You then create program logic for the events associated with the user interface components; the GUI Code Assistant helps you create your program logic.

COBOL for OS/2

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.

VisualAge COBOL Tools

Although the IBM COBOL language is practically the same across platforms, there are some minor differences between IBM COBOL for MVS & VM and VisualAge COBOL. These differences are documented in the *Summary of Differences: Host COBOL and Workstation COBOL* topics in the *IBM VisualAge for COBOL for OS/2 Programming Guide* and the *IBM COBOL Language Reference*. Porting considerations are described in “Porting Applications between Platforms” in the *IBM VisualAge for COBOL for OS/2 Programming Guide*.

Object-oriented Extensions

VisualAge COBOL's object-oriented (OO) language extensions are based on the emerging ANSI OO COBOL standard and are a syntax extension to COBOL. These are the same OO extensions you get with IBM COBOL for MVS. These extensions implement a complete OO paradigm that allows 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 OO languages enabled for IBM's System Object Model (SOM). This is provided through VisualAge COBOL's Direct-to-SOM capability.

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 local VSAM APIs
- Copy, sort, and merge functions for both record and byte files
- General data conversion APIs and services for both character and numeric data.

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.

Interactive Debugger

The Interactive Debugger for OS/2 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 until the return statement of the current program. The

VisualAge COBOL Tools

Interactive Debugger can also step over any program for which debugging is not available, for example, library and system routines.

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.

View the Program Source Code

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

Performance Analyzer

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

It enables you to monitor your program as it runs and generate a function-by-function trace of the run. The 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 debuggers, 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.

VisualAge COBOL Tools

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.

VisualAge COBOL Tools

Tasks and Information for VisualAge COBOL

Tasks and Information for VisualAge COBOL

The following table maps tasks you can perform with VisualAge COBOL to the steps for beginning the task and the information you need to continue the task. Any menu choices referenced are those of a default COBOL project.

To do this	Here's how to start	For more information, see
Install the products	Insert the VisualAge COBOL CD-ROM in your CD-ROM drive, then follow the installation instructions.	"Installing VisualAge COBOL" on page 1
Learn about VisualAge COBOL	Double-click on the Information Notebook icon in the VisualAge COBOL folder. Double-click on a topic's icon to view information on the topic.	"Your Next Step for Learning VisualAge COBOL" on page 79
Create a COBOL project without a GUI	Open the VisualAge COBOL folder and double-click on the Create New Project icon. Select the Create a default COBOL project radio button and complete the information in the window. Click on OK to create the project.	Task Helper Create New Project online help
Create a COBOL project with a GUI	Open the VisualAge COBOL folder and double-click on the Create New Project icon. Select the Select from predefined project types radio button and click on the OK push button. From the COBOL Project Smarts catalog, select COBOL GUI Designer Project and click on the Create push button.	Task Helper Create New Project online help
Work with an existing COBOL project	Double-click on the project folder. Use the project view menu-bar choices or the pop-up menu on the project parts to work with the project.	How Do I...? help in the project view window Online help in the project view window
Work with an existing COBOL GUI project	Double-click on the project folder. From the project view, select Project → Edit → GUI Project Edit . When the COBOL GUI Designer window opens, click on the icon you want to edit in the tree view (for example, the window part) and select Selected → Open .	How Do I...? help in the project view window COBOL GUI Designer online help
Create a new COBOL source file	From a COBOL project window, select Project → Create → Create New Text File with the COBOL Editor . When the New window appears, click on the drop-down arrow next to the Language Profile drop-down combo box. Scroll to locate CBL , then click on CBL to select the COBOL language-sensitive editing. Click on the New push button. An untitled file is created.	Task Helper <i>IBM VisualAge for COBOL for OS/2 Programming Guide</i> <i>IBM COBOL Language Reference Manual</i>

Tasks and Information for VisualAge COBOL

To do this	Here's how to start	For more information, see
Setting up a project with MVS files	<p>This task requires having the BETA level Remote Edit/Compile component installed and a working connection to the host with correctly configured APPC. Read the topics "Setting Up Communications" and "Setting Up Your Project" in the Task Helper before performing this task.</p> <p>Double-click on the Create New Project icon. In the Create New Project window, select the Select from predefined projects radio button and click on OK. Select COBOL MVS Project from the predefined projects list. Complete the dialogs, ensuring that you specify the PDS members when prompted.</p> <p>See the Task Helper for further information on proper PDS naming, connecting to the server, and adding additional PDS members.</p>	<p>Task Helper</p> <p>Appendix C, "Configuring APPC Communications" on page 149</p>
Create a COBOL program with a GUI	<p>Create a COBOL project with a GUI. When the COBOL GUI Designer appears, use it to create your GUI and the supporting logic.</p>	<p>Task Helper</p> <p>COBOL GUI Designer online help</p>
Set compiler options	<p>From a COBOL Project window, select Options→Compile. Complete the choices in the notebook.</p>	<p><i>IBM VisualAge for COBOL for OS/2 Programming Guide</i></p> <p>Task Helper</p> <p>Online help in the project view window</p>
Build a COBOL project (without a GUI)	<p>Create a COBOL source file, and save it with the extension cb1 (for example, mypgm.cb1). Build the program by selecting Project→Build→Build normal.</p>	<p>Task Helper</p> <p>How Do I...? help in the project view window</p> <p>Online help in the project view window</p>
Build a COBOL project (with a GUI)	<p>Create a GUI COBOL program. Build the program from the COBOL GUI Designer window by selecting Project→Build.</p>	<p>Task Helper</p> <p>How Do I...? help in the project view window</p> <p>COBOL GUI Designer online help</p>
Debug and test your programs	<p>Set the compiler options for debugging. On the compiler options notebook Debug page, click on the Compiler generates debugging information and Linker includes debugging information check boxes. Compile and link your program. From the COBOL project menu, select Debug.</p>	<p>"Using the Interactive Debugger" on page 110</p> <p>Task Helper</p> <p>Debugger online help</p> <p>Online version of the <i>Debug Tool User's Guide and Reference</i> in the Information Notebook</p>

Tasks and Information for VisualAge COBOL

To do this	Here's how to start	For more information, see
Analyze program performance	Set the compiler options for program analysis. On the compiler options Other page, click on the Produce profiling information check box. Compile and link your program. From the COBOL project icon view, click on the exe file with your right mouse button. Select Analyze from the pop-up menu.	Task Helper Performance Analyzer online help
Add an action	Select View → Tools setup . When the Tools setup window appears, ensure that you are viewing the Action view. The title "Actions" appears below the toolbar, and a tree view of the project's actions appears in the window. Select Actions → Add .	"Create an action" topic in the project view How Do I...? help Online help in the project view window
Create SQL statements	First, create a data structure mapping for your program (refer to the next task below). When you are coding your program in the COBOL Editor window, select Edit → Insert code → SQL to access the SQL Construction view .	"Creating SQL Statements with Data Assistant" on page 117 Task Helper How Do I...? help for Data Assistant Data Assistant online help
Create a data structure mapping	Creating a data structure involves dragging tables from the Database Schema view and dropping them into the Data Structure Mapping view . Start DB2 for OS/2, the Schema view , then the Mapping view from your project: <ul style="list-style-type: none"> • Select Project→Data Tools→DB2 Start. When prompted, enter your user ID and password. • Select Project→Data Tools→Data Assistant Schema view. When prompted, enter the database name. • Select Project→Data Tools→Data Assistant Mapping view. When prompted, enter a mapping file name. Follow the instructions in the online help to perform the needed data structure mapping.	"Creating SQL Statements with Data Assistant" on page 117 Task Helper How do I...? help for Data Assistant Data Assistant online help
Create CICS ECI calls	When coding your program in the COBOL Editor window, select Edit → Insert code → CICS ECI . Specify your parameters in the Transaction Assistant window. The generated code issues a COBOL CALL to ECICALL, which performs the ECI call with the parameters you specify.	"Using the CICS Transaction Assistant" on page 127 Task Helper Transaction Assistant online help
Access VSAM files (remote file access)	This is an advanced topic. Double-click on the Task Helper icon in the VisualAge COBOL folder for information on how to perform this task. The VSAM information is included in the different Task Helper paths, as appropriate to the task.	Task Helper <i>VSAM in a Distributed Environment</i>

Tasks and Information for VisualAge COBOL

To do this	Here's how to start	For more information, see
Write client/server applications	This is an advanced topic. Double-click on the Task Helper icon in the VisualAge COBOL folder for information on how to perform this task. You might need to view the introductory information for all paths presented. For example, you could use a COBOL non-GUI program as the server and a GUI program as the client.	Task Helper
Write object-oriented applications.	This is an advanced topic. Double-click on the Task Helper icon in the VisualAge COBOL folder for information on how to perform this task. Any separate considerations for object-oriented programming are noted in the different Task Helper paths.	Task Helper <i>IBM VisualAge for COBOL for OS/2 Programming Guide</i> <i>IBM COBOL Language Reference</i>
Print VisualAge COBOL publications	The VisualAge COBOL publications are included as PostScript files with the product. The files are located in the \PRINTABL directory on the VisualAge COBOL CD-ROM. Use your site's procedures for printing these files.	"Printing Publications" on page 81
Understand warranty information	See the enclosures in the IBM VisualAge for COBOL for OS/2 product box.	<i>IBM VisualAge for COBOL for OS/2 License Information</i> <i>Program License Agreement</i>

Using VisualAge COBOL Information

Using the Information with VisualAge COBOL

You can get information from many sources while using VisualAge COBOL. Besides this guide, there is:

- *IBM VisualAge for COBOL for OS/2 Programming Guide*, which helps you create, compile, link, and run VisualAge COBOL application programs
- *IBM COBOL Language Reference*, which provides you with the IBM COBOL language syntax
- **Task Helper** (located in the **VisualAge COBOL** folder), an online document that guides you in performing VisualAge COBOL application development tasks
- **Information Notebook** (located in the **VisualAge COBOL** folder), an online notebook from which you can view VisualAge COBOL information
- Online help, which gives you help from within the VisualAge COBOL tools
- **How Do I...?** help, which provides you with instructions on how to perform project-related tasks. It is located in the **Information Notebook** and from the VisualAge COBOL tools' **Help** menu-bar choices.

Your Next Step for Learning VisualAge COBOL

If you have completed “Build Your First VisualAge COBOL Application” on page 43 or “Build Your First VisualAge COBOL GUI Application” on page 49, you already have a taste for using VisualAge COBOL.

After reading this part of this book, “Getting Started with VisualAge COBOL,” continue to the next part, “VisualAge COBOL Tutorials.” The tutorial in that section will provide a more in-depth sample application for you to develop.

After you feel comfortable with the material in the tutorial, you can use the online **Task Helper** to give you information on creating different types of applications, depending on the type of application you want to develop. VisualAge COBOL provides sample applications for various types of applications that you can develop. These samples are located in the **VisualAge COBOL Samples** folder. Descriptions of the samples and instructions on how to build and run the Employee Lookup applications are included in **Samples Information** in the **VisualAge COBOL Samples** folder as well as in Appendix B, “VisualAge COBOL Supplied Sample Applications” on page 139.

Once you've started using VisualAge COBOL to develop your own applications, online help and online reference information can help you find specific information. Each of the tools has its own online help, which enables you to find specific information about using the tools. You can access the online guides and references for VisualAge COBOL from the **Information Notebook**.

Using VisualAge COBOL Information

Using the Online Reference Information

The easiest way to access the online reference information that comes with VisualAge COBOL is to use the **Information Notebook**. From the **Information Notebook**, you can click on an icon in the notebook to view an online document. Once you are viewing the online document, you can use the viewing program to search for information or to print information.

Using the Online Help

Information on how to use VisualAge COBOL tools is available through online help. You can access different kinds of help in one of the following ways:

- Select an item from a **Help** pop-up or menu bar choice in any VisualAge COBOL window. The **Help** menu-bar choice in a window gives you access to the forms of help available for the tool you are using. You can also select the **Help index** menu choice to view an list of the topics.
- Highlight a graphical user interface control (for example, a menu item) and press **F1** to get help on the control.
- To get general help on a window, select the **Help** menu-bar choice, then select **General Help**
- Click on the **Help** push button, where available.

Locating Online Help Topics

When using online help, sometimes you may not find the help you need right away. There are several ways to find the information you want.

Using Hypertext

Within some help windows, certain words or phrases are highlighted. To obtain additional information about a highlighted word or phrase, double-click on the word or phrase. You can also press the **Tab** key until the cursor is on the word or phrase (reverse video by default), and then press **Enter**.

Using the Help Index and Contents

The Help Index and the Contents display the list of available help windows.

To Use the Help Index: From an VisualAge COBOL tool window, select the **Help** menu-bar choice, then select **Help index**. You can also see the help index by pressing **F11** or clicking on the **Index** push button while any help window is open.

If you want to view an index topic, double-click on the topic. After you select a topic, the help information for that topic appears in a window.

To Use the Contents: From a help window, select the **Options** menu, then select **Contents**. The Contents window appears. A plus sign (“+”) next to a topic indicates that help is available for sub-topics related to that topic. Click on the plus sign to see the complete list.

Using VisualAge COBOL Information

If you want to view a topic, double-click on the topic. After you select a topic, the help information for that topic appears in a window.

Using Search

To search for a topic in an VisualAge COBOL tool's online help:

1. Select **Search** from the **Services** menu.
2. In the **Search for** field, type the word or phrase you want to search for.
3. Select one radio button to indicate where you want to look for the word or phrase.
4. Select **Search**. If the word or phrase is not found, a message window is displayed.
5. If your word or phrase is found, a search window appears with a list of topics in which the word or phrase is found. Choose a topic you would like to view, then double-click on the topic.

Printing Online Help Topics

To print online help topics, either click on the **Print** push button, where available, or select **Services**→**Print**.

When the Print window appears, select what you would like to print and click on the **Print** push button. For example, to print the section you are viewing, select the **This section** radio button and click on **Print**.

Getting VisualAge COBOL Publications

All of the VisualAge COBOL publications are included as viewable softcopy files in the Information Notebook and as printable files.

Printing Publications

The VisualAge COBOL publications are included with the product as printable, formatted PostScript** (.PS) files. They are located in the \PRINTABL directory on the VisualAge COBOL product CD-ROM.

To print these files, follow the procedure for your installation for printing PostScript files. A typical example for OS/2 with TCP/IP would be to enter the command `lpr filename`, where `filename` is the name of the file you want to print. You could also drag the file from the **PRINTABL** folder on the VisualAge COBOL CD-ROM and drop it on the PostScript printer icon, if your system is set up for this procedure.

Ordering Publications

You can order printed copies of these publications using one of the following methods:

- If you are a customer in the United States, call IBM Software Manufacturing Solutions at 1-800-879-2755. You can also fax a request to the National Publication Order Center at 1-800-445-9269.
- If you are an international customer, contact your IBM Authorized Dealer or your IBM Marketing Representative.

Using VisualAge COBOL Information

Table 2. Publications for the COBOL Family of Products

COBOL Product	Book Title	Order Number
IBM VisualAge for COBOL for OS/2	Licensing	n/a
	Getting Started	GC26-8421
	IBM COBOL Language Reference	SC26-4769
	Programming Guide	SC26-8419
	VSAM in a Distributed Environment	SC26-7063
	Data Description and Conversion	SC26-7091
	Data Description and Conversion: A Data Language Reference	SC26-7092
IBM COBOL Set for AIX	SMARTsort for OS/2 and AIX	SC26-7099
	Licensing	n/a
	Getting Started	GC26-8425
	IBM COBOL Language Reference	SC26-4769
	Programming Guide	SC26-8423
	Program Builder User's Guide	SC09-2201
	LPEX User's Guide and Reference	SC09-2202
	VSAM in a Distributed Environment	SC26-7064
	Data Description and Conversion	SC26-7066
	Data Description and Conversion: A Data Language Reference	SC26-7092
SMARTsort for OS/2 and AIX	SC26-7099	
IBM COBOL for MVS & VM	Compiler and Run-Time Migration Guide	GC26-4764
	Installation and Customization under MVS	SC26-4766
	Licensed Program Specifications	GC26-4761
	Programming Guide	SC26-4767
	Diagnosis Guide	SC26-3138
	IBM COBOL Language Reference	SC26-4769

You can also order books by the set. These are included in a bill-of-forms (BOF).

IBM VisualAge for COBOL for OS/2 (SBOF-7333) - Includes:

Language Reference (SC26-4769)

Programming Guide (SC26-8419)

SMARTdata UTILITIES for OS/2 (SBOF-6131) - Includes:

VSAM in a Distributed Environment (SC26-7063)

Data Description and Conversion (SC26-7091)

Data Description and Conversion: A Data Language Reference (SC26-7092)

SMARTsort for OS/2 and AIX (SC26-7099)

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 Started Period

After purchasing the VisualAge COBOL product, you, or the designated contacts in your establishment, can receive free support or unlimited voice support calls to IBM that is related to usage, setup, or installation. This support starts with the **first** phone call to IBM and lasts **60 days**. Call 1-800-237-5511 or 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.

At the conclusion of the Getting Started period, continued voice support is available for a fee. Multiple voice support options are also available for a fee.

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:** 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 76711.611@CompuServe.com.

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

- **Fixes:** You can access fixes from the following sources.
 - Download the fixes from the FTP site at:
`ftp://ftp.software.ibm.com/ps/products`
 - 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. Scroll to the **Support** section.
 3. Click on **...technical support**.

VisualAge COBOL Support

4. From the **IBM COBOL Support Services** page, scroll down to and click on **fixes** for VisualAge COBOL.
 - Call 1-800-237-5511 to request packaged fixes in the form of a CD-ROM. There is a fee associated with the CD-ROM.
- **Mail:** Mail your comments to:
 - IBM Corporation
 - Personal Systems Support Family
 - Internal Zip 2901
 - 11400 Burnett Road
 - Austin, Texas 78758
- **World Wide Web:** 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. Scroll to the **Support** section for information about technical support available.
- **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. Scroll to the **Services and Education** section for information about IBM's consulting services.

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

VisualAge COBOL Support

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. Scroll to the **Services and Education** section for information about the various education offerings by IBM.

VisualAge COBOL Support

VisualAge COBOL Tutorials

This section contains more complicated hands-on tutorials.

In “Creating a Tax Computation Application with a GUI,” you will build an application, including the subroutine that the application calls, using the VisualAge COBOL tools. When you are finished creating both the subroutine and application projects, you will nest the subroutine project inside the application project.

In “Creating SQL Statements with Data Assistant,” you will graphically view your relational database. Data Assistant allows you to map COBOL data structures to the database and generates SQL statements into your source file.

In “Using the CICS Transaction Assistant,” you will generate a CICS ECI call and parameter list for invoking CICS transactions. Transaction Assistant simplifies the task of constructing CICS transaction calls in COBOL programs.

“VisualAge COBOL Tutorials” assumes that you have gone through the instructions in “Getting Started with VisualAge COBOL” on page 35.

The topics in this section are:

Creating a Tax Computation Application with a GUI	89
Creating SQL Statements with Data Assistant	117
Using the CICS Transaction Assistant	127

Creating an Application with a GUI

Creating a Tax Computation Application with a GUI

This chapter guides you through building an application, including the subroutine that the application calls. The application name is *Tax Computation* and it contains a graphical user interface (GUI). The user enters a sales amount and presses a push button. The application calculates the tax for the sales amount and displays the total.

The tax is calculated by the subroutine, *Tax Calculation*. This subroutine's parameter list is defined as follows:

```
01 TAXCALCU-PARM-LIST.  
   05 SALES-AMOUNT-CHAR PIC X(5).  
   05 TOTAL-AMOUNT-CHAR PIC X(8).
```

SALES-AMOUNT-CHAR is the amount the user enters. TOTAL-AMOUNT-CHAR is the total amount, including the tax, the application will display.

When you finish, you will have an application that displays the total amount with sales tax included. Figure 35 shows you what the application's interface will look like when you have finished.

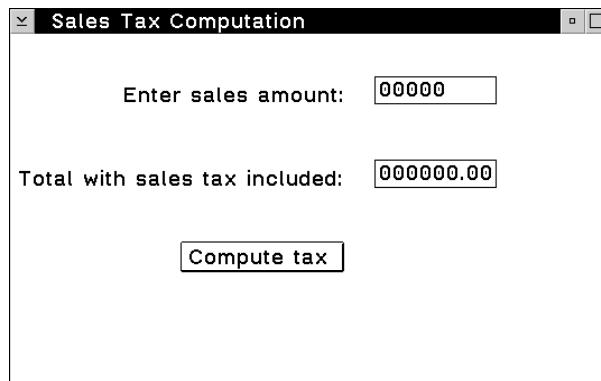


Figure 35. *Tax Computation Application*. A COBOL application you will be building using VisualAge COBOL.

The main steps you will follow are:

- Creating the subroutine project
- Creating the subroutine logic
- Creating the GUI application project
- Creating the application logic
- Nesting the projects
- Setting compiler options
- Building the application
- Debugging the application
- Running the application
- Packaging the application for distribution

Creating the Subroutine Project

Creating the Subroutine Project

Your first step in developing an application with VisualAge COBOL is to set up a project. A project encapsulates all of the components you need to build a single target. For example, the Tax Calculation subroutine project you create in this chapter contains a COBOL source file (a component or project part) from which you build the subroutine object file (a target).

To set up a new project:

1. Double-click on the **VisualAge COBOL** icon. The VisualAge COBOL - Icon View window opens.
2. Double-click on the **Create New Project** icon. The IBM VisualAge COBOL - Create New Project window opens.

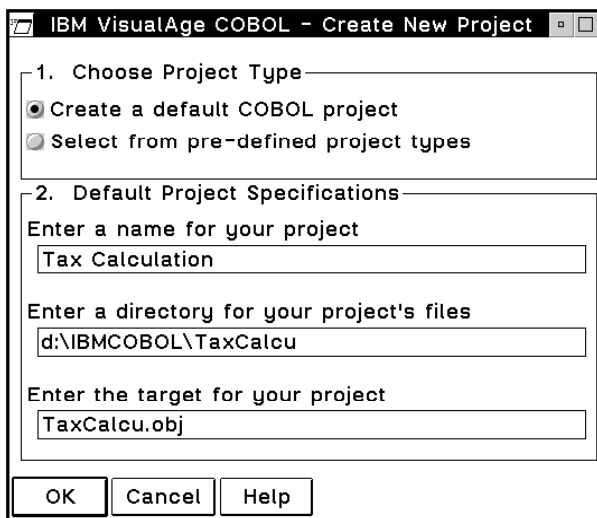


Figure 36. Create New Project window. The starting point for creating COBOL projects.

3. In the **Choose Project Type** group, ensure that the **Create a default COBOL project** radio button is selected.
4. In the **Default Project Specifications** group, specify your project's name and the location where the project's files are to be stored. In the entry field titled **Enter a name for your project**, enter the name Tax Calculation.

Note: For file and directory names, you can use up to 8 alphanumeric characters and a dash (-).

5. Click with mouse button 1 in the entry field titled **Enter a directory for your project's files**. Notice that this entry field and the entry field below it are updated automatically to reflect the project name you entered.

Creating the Subroutine Logic

6. In the **Enter the target for your project** entry field, specify your project's target. For the Tax Calculation subroutine, change `taxcalcu.exe` to `taxcalcu.obj`. The subroutine's object file (`taxcalcu.obj`) will be created when you build (or compile) this project. This object file, in turn, will be statically linked to the main application.
7. Click on the **OK** push button to create the project. Since the directory for the Tax Calculation (`\TAXCALCU`) has not yet been created, a message window appears. Click on the **OK** push button. The directory is created and the new project opens.

Now that you have created your project, you are ready to create the files you need for the application.

Creating the Subroutine Logic

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

To create the COBOL source file for the Tax Calculation subroutine project:

1. From the **Tax Calculation - Icon view** window, select the **Project** menu bar choice, then select the arrow button next to the **Create** choice.
2. A cascaded menu appears. From the cascaded menu, select **Create New Text File with the COBOL Editor**.

The **New** window appears. In this window, you can specify which language sensitive editing features you want. Ensure that the **Language profile** check box is checked. The entry field for the **Language profile** list box is initially blank. Scroll through it and select **CBL**. This sets the editor to be COBOL language sensitive.

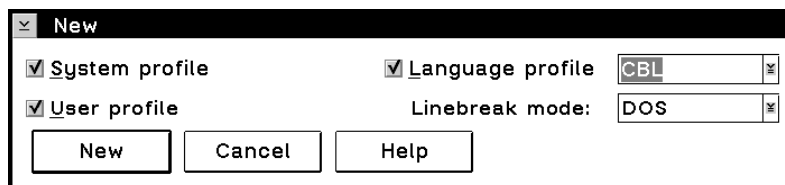


Figure 37. New window

- Click on **New**.
3. The COBOL Editor displays the window titled **Editor - Untitled Document 1**.
4. In the editor window, starting at column 8 (under A), type the source code shown in Figure 38 on page 92.

Creating the Subroutine Logic

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    TAXCALCU.  
AUTHOR.       Programmer.  
  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 PROGRAM-CONSTANT-FIELDS.  
   05 SALES-TAX          PIC V9(2) VALUE .08.  
  
LINKAGE SECTION.  
  
01 TAXCALCU-PARM-LIST.  
   05 SALES-AMOUNT-CHAR PIC X(5).  
   05 SALES-AMOUNT      REDEFINES SALES-AMOUNT-CHAR  
                        PIC 9(5).  
   05 TOTAL-AMOUNT-CHAR PIC X(8).  
   05 TOTAL-AMOUNT      REDEFINES TOTAL-AMOUNT-CHAR  
                        PIC 9(6)V9(2).  
  
PROCEDURE DIVISION USING TAXCALCU-PARM-LIST.  
  
   COMPUTE TOTAL-AMOUNT = SALES-AMOUNT +  
     (SALES-AMOUNT * SALES-TAX).  
  
   GOBACK.
```

Figure 38. COBOL Source Code for the Tax Calculation Subroutine

5. When you have finished entering the source code, save the file. To save the file, select the **File** menu bar choice, then select **Save**. The **Save as** window appears. Ensure that the **Directory** list box shows the directory you specified in step 5 on page 90 (for example, IBMCOBOL\TAXCALCU). In the **Save as filename** field, type `taxcalcu.cb1` and click on the **Save As** push button.
6. The COBOL Editor is in view. Close it by double-clicking on its system menu symbol.

Creating the GUI Application Project

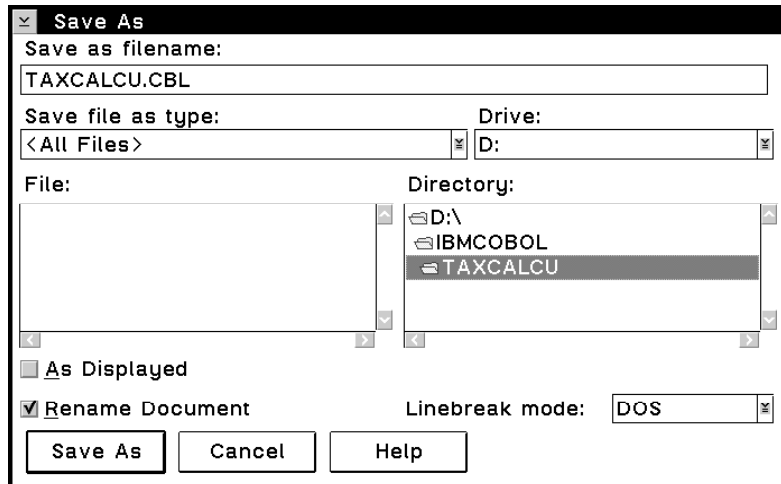


Figure 39. Save as window

7. If the file does not appear in the **Tax Calculation - Icon view** window, make the window active by clicking on its title bar, then press **F5** to refresh the view.
8. Close the **Tax Calculation - Icon view** window by double-clicking on the system menu symbol in the upper left corner of the window.

Creating the GUI Application Project

Your first step in developing a GUI application with VisualAge COBOL is to set up a GUI project. Like other projects, the GUI project contains the parts you need to create a specific target; for example, the files you need to create an application that has a GUI.

The **Project** menu bar choice in the COBOL GUI Designer window provides access to project-level menu actions that will help you edit, compile, and debug the software you develop.

To set up a new GUI project:

1. Double-click on the **Create New Project** icon. The **IBM VisualAge COBOL - Create New Project** window opens in the VisualAge COBOL - Icon View window.
2. In the **Choose Project Type** group, select the **Select from predefined project types** radio button and click on the **OK** push button.
3. The COBOL Project Smarts - Catalog View window opens. From the **Available Projects** list box, select **COBOL GUI Designer Project** and click on the **Create** push button.
4. After a few moments, the COBOL GUI Designer, Parts Palette, and Window with canvas windows open.

Customizing the GUI

Creating the Graphical User Interface (GUI)

You are ready to create the GUI for the application. You will use the COBOL GUI Designer to create the GUI for the Tax Computation application.

Since you chose to create a GUI Project during the project set-up, the COBOL GUI Designer - Untitled, Parts Palette, and Window with canvas windows should appear on your desktop.

Adding the GUI Parts

Using the Parts Palette or the Parts Catalog, drag and drop the following parts onto the **Window with canvas**. When you point your mouse on a part, the name of the part is displayed at the bottom of the Parts Palette window.

As you place the parts on the Window with canvas, make sure that you align your parts with the *bottom* margin of the window. This ensures that if you change the size of the window, your parts will stay in view.

1. Drag and drop two **Static Text** fields onto the window.
2. Drag and drop two **Entry** fields onto the window. The first should be between the first static text field and the right border of the window. The second should be beneath the first entry field.
3. Drag and drop one **Push button** below the second static text field at the bottom of the window.

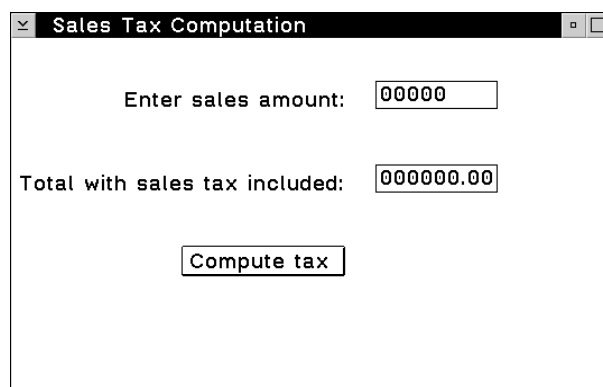


Figure 40. Tax Computation Application

Customizing the GUI

When you have parts on the window, you can change the parts to customize them for your application.

For the Tax Computation application, you need to change some of the part attributes. You can change these attributes by using the settings notebook.

Saving the GUI Project

The **settings notebook** is used to set the initial run-time attributes for parts on your interface, such as color and font. You can also use the settings notebook for a part to change the part's name, which your program uses to query and change the part's appearance while the program runs.

First, change the attributes for the first entry field part:

1. Double-click on the first **Entry** field part to open the **settings notebook**.
2. Move the cursor to the **Part Name** entry field and change the contents to ENT-SALESAMOUNT.

The part name is the internal name by which your part will be known. This name will also appear in your COBOL program and in the VisualAge COBOL tools that assist you in creating the COBOL code. You don't have to change the part name. By giving it a meaningful name, as you would with a variable in a program, the part will be easier to identify when you are creating the Tax Computation Application logic. to the correct GUI part.

3. Select the **Data** tab and set the **Length** of the entry field to 5.
4. Change the **Type** to **Numeric**.
5. Ensure the **Decimals** field is set to 0.
6. Double-click on the system menu symbol at the upper left corner of the settings notebook window. Your changes are saved.

Next, change the attributes for the rest of your parts:

- For the second entry field part:
 - Change the part name to ENT-TOTAL.
 - Select the **Style** tab and click on the **Read-only** check box to select it.
 - Select the **Data** tab and change the **Length** to 8, the **Type** to **Numeric**, and the **Decimals** to 2.
- For the push button, change the part name to PSB-COMPUTETAX and the label to Compute tax.
- For the first static text part, change the text to Enter sales amount:.
- For the second static text, change the text to Total with sales tax included:.
- For the Window with canvas part, double-click on the **title bar** of the Window with canvas. Change the part name to FRA-TAXCOMPUTATION and the title to Sales Tax Computation.

To move and size the GUI parts, see “Moving and Sizing GUI Parts” on page 55.

Saving the GUI Project

To save the GUI project, select **Project**, then select **Save** from the **COBOL GUI Designer - Untitled** window.

Saving the GUI Project

The **COBOL GUI Designer - Save as Application** window appears. When you save a new application, the COBOL GUI Designer - Save as Application window enables you to select the name and location of your project.

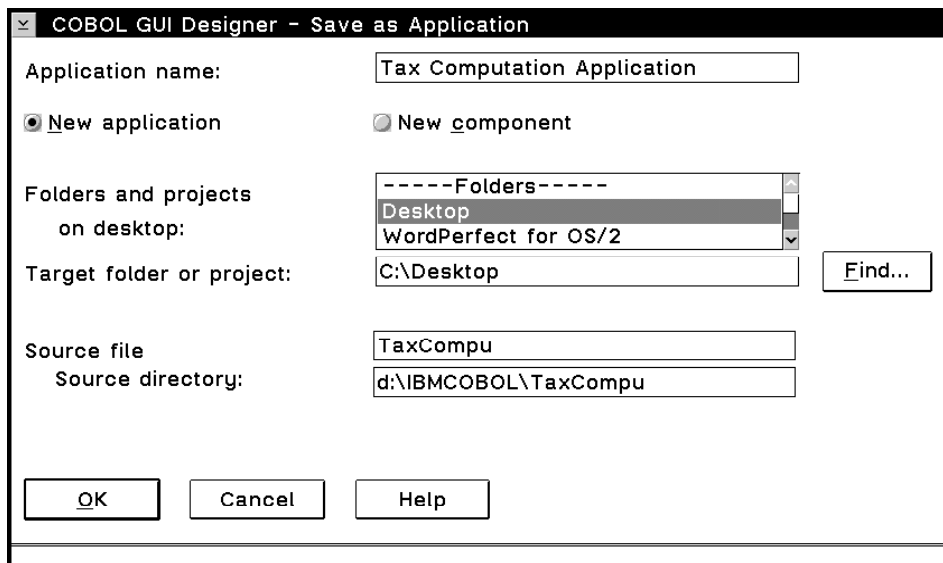


Figure 41. COBOL GUI Designer - Save as Application window

For the Tax Computation Application, perform the following steps:

1. Type Tax Computation Application in the **Application Name** field.
2. Ensure that the **New application** radio button is selected. It is the default.
3. For **Folders and projects on desktop**, select the **Desktop** item.
4. Click in the **Source file** field. TAXCOMPU is automatically displayed in the Source file and Source directory entry fields. This name is generated based on the application name you specify. In this example, it is based on the name Tax Computation Application. You can change this name.
5. In the **Source directory** field, the default directory displayed is a modification of the current directory. For example, if you are in the directory D:\IBMCOBOL and your source file name is TaxCompu, the default source directory is D:\IBMCOBOL\TaxCompu. You can change the default as desired.
Note: For file and directory names, you can use up to 8 alphanumeric characters and a dash (-).
6. Click on **OK**.

The Tax Computation Application project is saved. The COBOL GUI Designer window title changes from Untitled to your new application name. For the Tax Computation application, it changes to Tax Computation Application. The GUI source files are saved in the path you specified in the Source directory field.

Creating the Application Logic

Creating the Application Logic

Your next step for developing the Tax Computation Application involves coding the logic behind the parts in your GUI. The COBOL program behind your GUI follows the *event-driven programming* paradigm. For more information about this paradigm, see “Event-Driven Programming” on page 57.

The following steps show you how to create logic for the Tax Computation Application. When a user types an amount—for example, 100—in the **Enter sales amount** entry field and presses the **Compute tax** push button, the total amount of \$108.00 appears in the **Total with sales tax included** entry field.

You will create the logic to do this. First, you will indicate what the event is that your logic will respond to—for the Tax Computation application, the logic responds to the **Compute tax** push button being pressed.

1. Move the mouse pointer to the **Compute tax** push button on the **Sales Tax Computation** window and press mouse button 2. A pop-up menu appears.
2. Select **Events**, then select **PRESS** from the cascaded menu. You will be specifying the logic that is to be run when the user clicks on (or presses) the **Compute tax** push button.

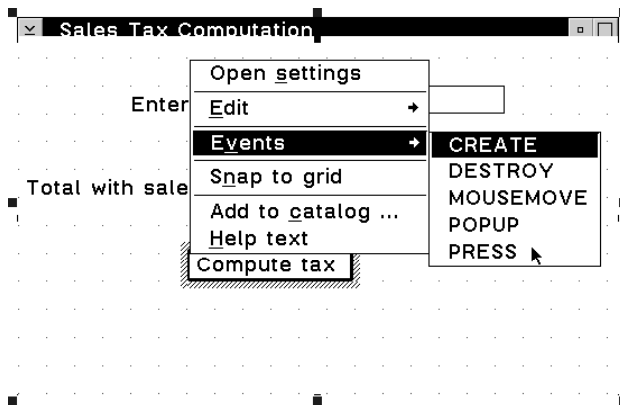


Figure 42. Selecting the PRESS event for the Compute tax push button

3. The COBOL Editor window opens at the ENTRY statement for the **Compute tax** push button press event.

Notice the format of the name on the ENTRY statement, FRA-TAXCOMPUTATION_PSB-COMPUTETAX_PRESS. The ENTRY name consists of the window name, the part name, and the name of the selected event, separated by underscores.

Creating the Application Logic

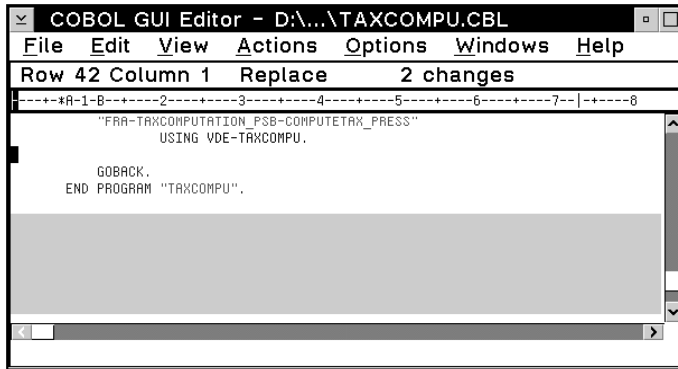


Figure 43. Editor window with ENTRY statement for the PRESS event

At this point, you need to specify what happens when the **Compute tax** push button is pressed. The application logic retrieves the contents of the **Enter sales amount** entry field, adds the sales tax, and moves the amount to the second entry field where it will be displayed for the user. IBM VisualAge for COBOL for OS/2 provides assistance to help you generate the COBOL code to work with the GUI parts.

1. Click on the title bar of the COBOL Editor window. The cursor in the window should appear on the line below the ENTRY statement (after USING VDE-TAXCOMPU). If the cursor is not positioned there, move it there now.

Select the **Edit** menu bar choice on the editor window, then select **Insert code**. From the cascaded menu, select **GUI**. The **GUI Code Assistant** window opens.

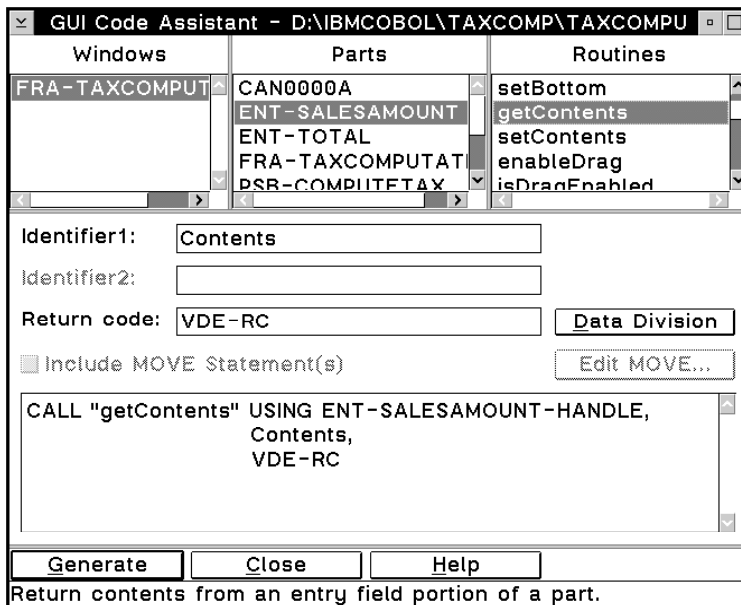


Figure 44. GUI Code Assistant window

Creating the Application Logic

2. Click on **FRA-TAXCOMPUTATION** in the **Windows** list box to select the window that contains the **ENT-SALESAMOUNT** field, the part from which you want to retrieve the contents.
3. Click on **ENT-SALESAMOUNT** in the **Parts** list box.
4. Click on **getContents** in the **Routines** list box to select the routine to get the value of the **Enter sales amount** entry field. You may need to scroll the **Routines** list box to find **getContents**.
5. Notice the CALL statement in the multiple-line entry field just above the push buttons at the bottom of the window. This is the statement that will be generated. Click on the **Generate** push button to complete.

A CALL statement is generated that gets the value of the **Enter sales amount** entry field. This CALL statement issues a call to the routine **getContents**, using parameters generated by the GUI Code Assistant. These parameters are based on the windows, parts, and routines you select.

Note: When you create code using the GUI Code Assistant window, you are using default variables. These variables are defined in a copy file, VACCESS.CPY, that is automatically included in this program. Many of the variables in this copy file are variable-length tables.

6. To close the GUI Code Assistant window, click on the **Close** push button.

The next action of this event is to move the value retrieved by the **getContents** routine from the Contents variable to the SALES-AMOUNT-CHAR variable.

1. Move the cursor in the COBOL Editor window to the line after the logic created to get the contents of the **SALESAMOUNT** field (CALL "getContents" USING ENT-SALESAMOUNT-HANDLE, Contents, VDE-RC).
2. Type the following code. To add extra lines, press the Enter key.

```
MOVE Contents-String(1:Contents-Length)
  TO SALES-AMOUNT-CHAR.
```

3. Because SALES-AMOUNT-CHAR is a new variable, you need to define it in the DATA DIVISION.

You can get to the DATA DIVISION either by scrolling up in your source code or by selecting **Data Division** from the **View** pull-down. The latter option opens another editor session with the cursor positioned at the DATA DIVISION. In the WORKING-STORAGE SECTION, add the following:

```
01 TAXCALCU-PARM-LIST.
   05 SALES-AMOUNT-CHAR PIC X(5).
```

4. Next, define the variable for the amount with sales tax computed by the subroutine. Also add the following line below the SALES-AMOUNT-CHAR variable:

```
05 TOTAL-AMOUNT-CHAR PIC X(8).
```

5. Return to your position in the PSB-COMPUTETAX_PRESS event by either scrolling back down, or if you used the **Data Division** option, by closing the editor session that is

Calling the Subroutine Logic

displaying the DATA DIVISION and returning to the original editor session. The two editor sessions are dynamic in that your changes show up in both sessions.

Calling the Subroutine Logic

1. Now, you need to write the code to call the subroutine. Below the MOVE statement, type:

```
CALL "TAXCALCU" USING TAXCALCU-PARM-LIST.
```

After the subroutine calculates the total with tax, it returns the value in the TOTAL-AMOUNT-CHAR variable, which you defined in your DATA DIVISION.

The final action is to display the value in the **Total with sales tax included** entry field. You can use the GUI Code Assistant window again to generate the code.

1. In the editor window, place the cursor on a blank line below the CALL "TAXCALCU" statement.
2. Select **Edit** from the COBOL Editor menu bar. Select **Insert code**, then select **GUI**.
3. When the GUI Code Assistant window appears, select **FRA-TAXCOMPUTATION** from the **Windows** list box and **ENT-TOTAL** from the **Parts** list box. Select the **setContents** from the **Routines** list box.
4. Ensure that the **Include MOVE Statement(s)** check box is checked.
5. Select the **Edit MOVE** push button. A separate window appears.

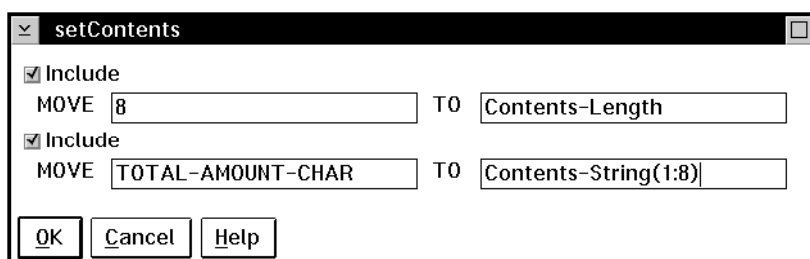


Figure 45. MOVE window

6. In the first MOVE statement, replace 1 with 8.
7. In the second MOVE statement, replace SPACES with TOTAL-AMOUNT-CHAR. Change Contents-String to Contents-String(1:8).
8. Click the **OK** push button.
9. The code in the multiple-line entry field should look like:

```
MOVE 8 TO Contents-Length
MOVE TOTAL-AMOUNT-CHAR TO Contents-String(1:8)
CALL "setContents" USING ENT-TOTAL-HANDLE,
    Contents,
    VDE-RC
```

Calling the Subroutine Logic

10. Click on **Generate**. Click on **Close** to close the **GUI Code Assistant** window.
11. You have now finished writing the code for the **Compute tax** event. The entire source code should look like the example in Figure 46.

```
PROCESS QUOTE LIB PGMNAME(MIXED) THREAD TRUNC(OPT)
IDENTIFICATION DIVISION.
PROGRAM-ID. "TAXCOMPU" RECURSIVE.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
* SOURCE-COMPUTER.
* OBJECT-COMPUTER.
SPECIAL-NAMES.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01 VDE-RC PIC S9(9) USAGE COMP-5.
   COPY RETURNCD.
   COPY VACCESS.
* LOCAL-STORAGE SECTION.
01 TAXCALCU-PARM-LIST.
   05 SALES-AMOUNT-CHAR PIC X(5).
   05 TOTAL-AMOUNT-CHAR PIC X(8).

LINKAGE SECTION.
COPY TAXCOMPU.
COPY VEVENTS.
```

Figure 46 (Part 1 of 2). COBOL Source Code for the Tax Computation Application

Calling the Subroutine Logic

```
PROCEDURE DIVISION USING CommandLine-Data.
  GOBACK.

  ENTRY
    "FRA-TAXCOMPUTATION_FRA-TAXCOMPUTATION_DESTROY"
      USING VDE-TAXCOMPU.
  MOVE VDE-TERMINATE-APPLICATION TO ACTION-RC.
  GOBACK.

  ENTRY
    "FRA-TAXCOMPUTATION_PSB-COMPUTETAX_PRESS"
      USING VDE-TAXCOMPU.

  CALL "getContents" USING ENT-SALESAMOUNT-HANDLE,
                        Contents,
                        VDE-RC

  MOVE Contents-String(1:Contents-Length)
    TO SALES-AMOUNT-CHAR.

  CALL "TAXCALCU" USING TAXCALCU-PARM-LIST.

  MOVE 8 TO Contents-Length
  MOVE TOTAL-AMOUNT-CHAR TO Contents-String(1:8)
  CALL "setContents" USING ENT-TOTAL-HANDLE,
                        Contents,
                        VDE-RC

  GOBACK.
END PROGRAM "TAXCOMPU".
```

Figure 46 (Part 2 of 2). COBOL Source Code for the Tax Computation Application

12. Save the source by selecting **Save** from the **File** pull-down. Close the Editor by double-clicking in the upper left corner of the window.

You just created the logic for the **Compute tax** push button.

The logic to close the GUI application has already been coded for you. To view this event logic:

1. Click mouse button 2 on the **Sales Tax Computation** title bar.
2. Select **Events**.

Notice the check mark on the **DESTROY** menu choice on the **Events** cascaded menu. This means that the event logic is already created. Clicking on the menu choice takes you to the existing event logic.

3. Select **DESTROY**.

The COBOL Editor window appears. The cursor is placed in the source code at the DESTROY event. This logic causes the application to end when the user closes the application's window.

Figure 47 on page 103 shows the COBOL Editor window with the event code you have added for the Tax Computation application.

Setting Compiler Options

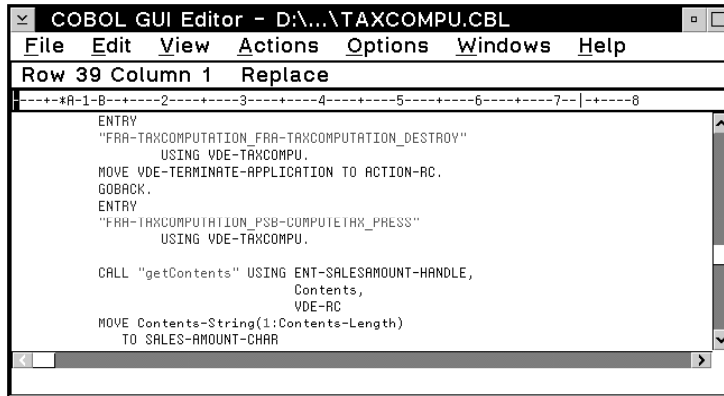


Figure 47. Event Logic for the Tax Computation Application

4. Close the COBOL Editor window.
5. Close the COBOL GUI Designer window.
6. The **COBOL GUI Designer - Save Project** window appears. Select **Save**.
7. Close the COBOL Project Smarts - Catalog View window.
8. Close the VisualAge COBOL - Icon View window.

Nesting the Projects

You can nest projects to reflect the calling structure. In this application, the Tax Computation Application calls the Tax Calculation subroutine. To reflect this calling structure, the Tax Calculation subroutine is nested in the Tax Computation Application GUI project.

The Tax Calculation subroutine you just created is a non-GUI project. You can nest this project in the Tax Computation Application.

1. Open the **Tax Computation Application** project by double-clicking on its icon.
2. Drag the **Tax Calculation** subroutine project icon from the desktop onto to an open area in the **Tax Computation Application** project window. This moves (nests) the subroutine project inside of the application project.

You will see the **Tax Calculation** subroutine project listed in the **Tax Computation Application - Icon view** window. This may take a few moments. You may need to press **F5** to refresh the view.

Setting Compiler Options

Now you will set the compile options for the Tax Calculation subroutine.

1. Open the **Tax Calculation** subroutine project by double-clicking on it in the **Tax Computation Application - Icon view** window.

Setting Compiler Options

2. Select **Compile** from the **Options** pull-down of the **Tax Calculation - Icon view** window. The **COBOL Compiler Options** notebook appears.

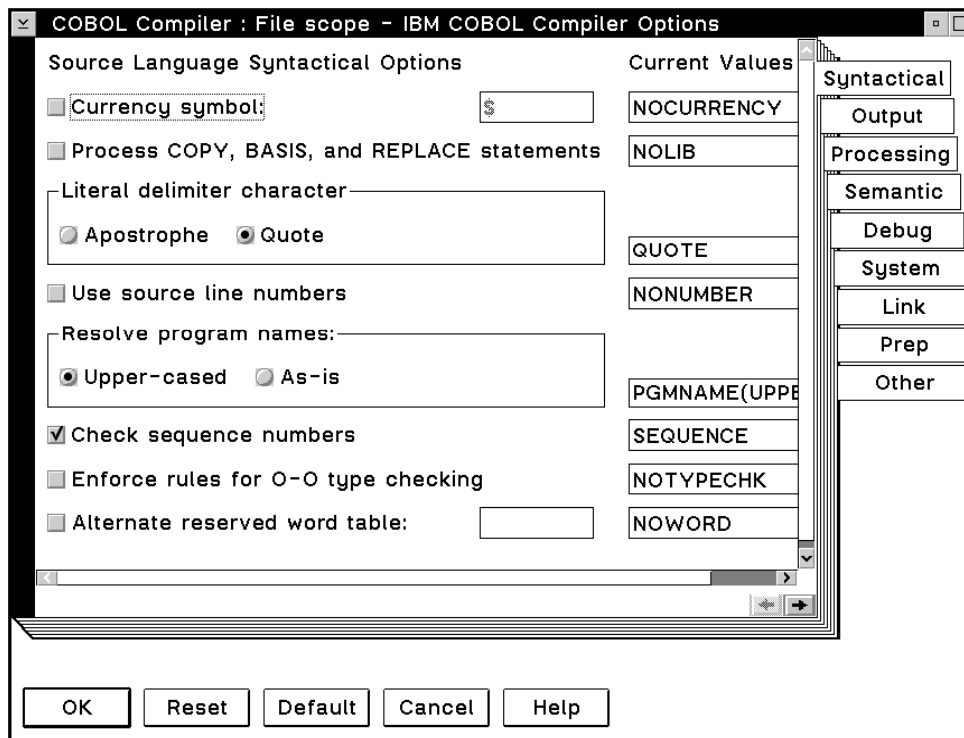


Figure 48. IBM COBOL Compiler Options notebook

3. Select the **Debug** tab to go to that page. Under the **Debugging information** group box, click on the **Compiler generates debugging information** and **Linker includes debugging information** check boxes.

Setting Compiler Options

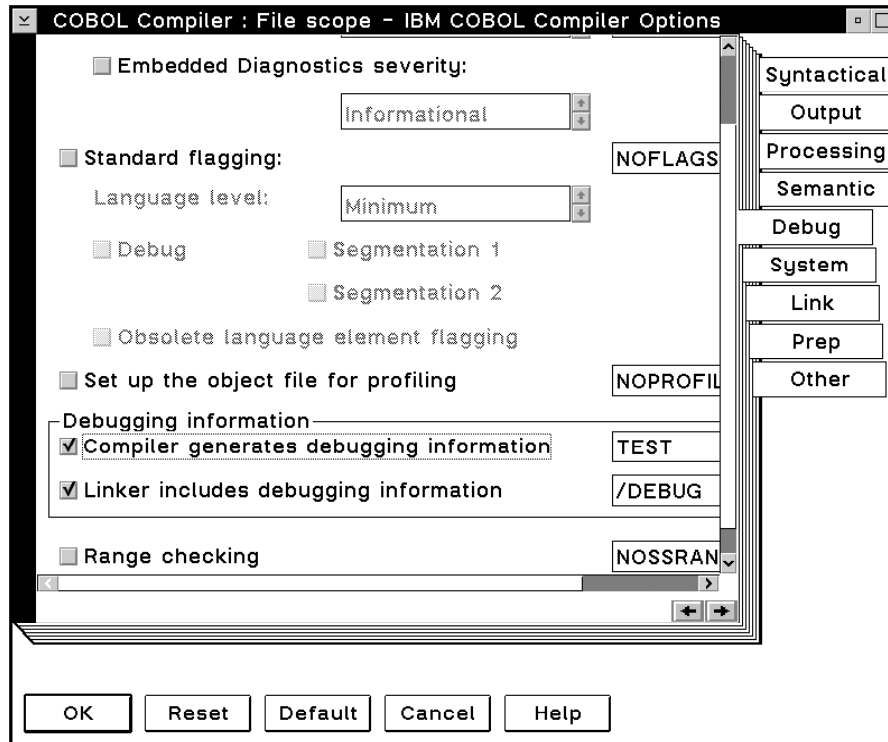


Figure 49. Setting compile options for the Tax Calculation subroutine

This step is done at this time so that you can debug the application at the source level later in this tutorial. It enables the compiler and linker to generate debugging information.

4. Click on the **Other** tab. Click on the **Compile programs but do not link** check box.

Setting Compiler Options

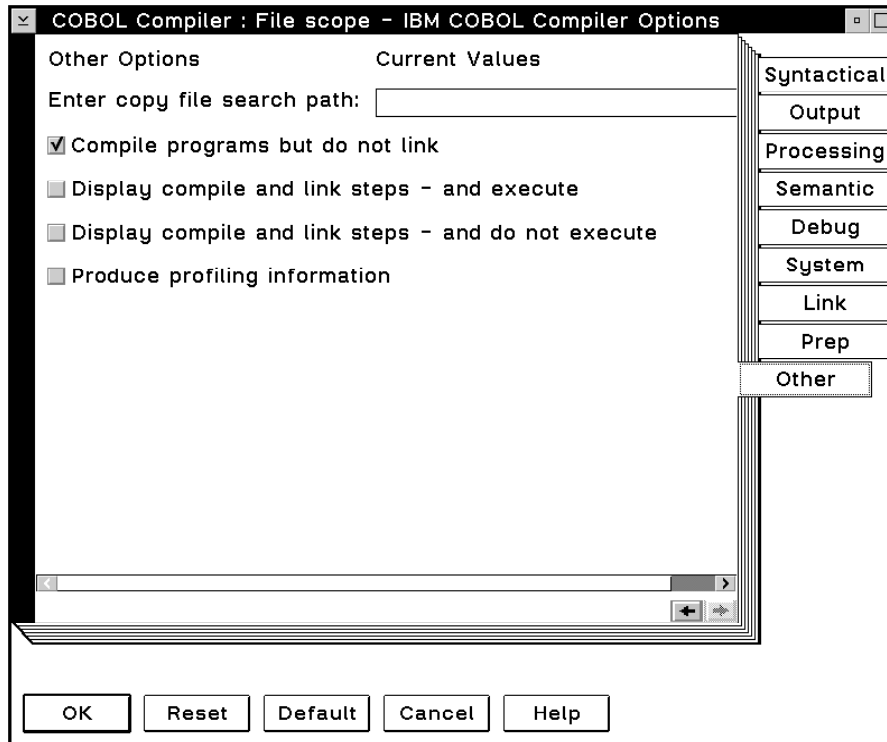


Figure 50. Setting compile options for the Tax Calculation subroutine

5. Click on **OK** to close the Compiler Options notebook.
6. Close the Tax Calculation - Icon view window.

Now you will set the compile options for the Tax Computation Application.

1. Select **COMPILE** from the **Options** pull-down of the **Tax Computation Application - Icon view** window. The **COBOL Compiler Options** notebook appears.
2. Select the **Debug** tab. Under the **Debugging information** group box, click on the **Compiler generates debugging information** and **Linker includes debugging information** check boxes. You may need to scroll down the notebook to see these check boxes.

This step is done at this time so that you can debug the application at the source level later in this tutorial. It enables the compiler and linker to generate debugging information.

Setting Compiler Options

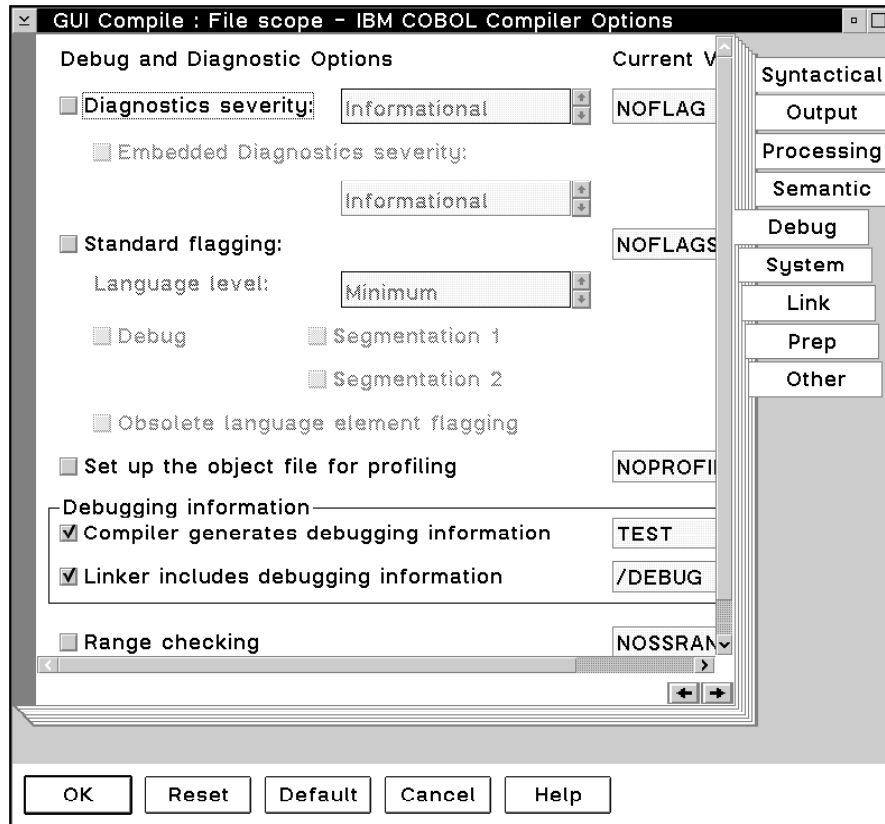


Figure 51. Setting compile options for the Tax Computation application

3. Select the **Link** tab.

Building the Application

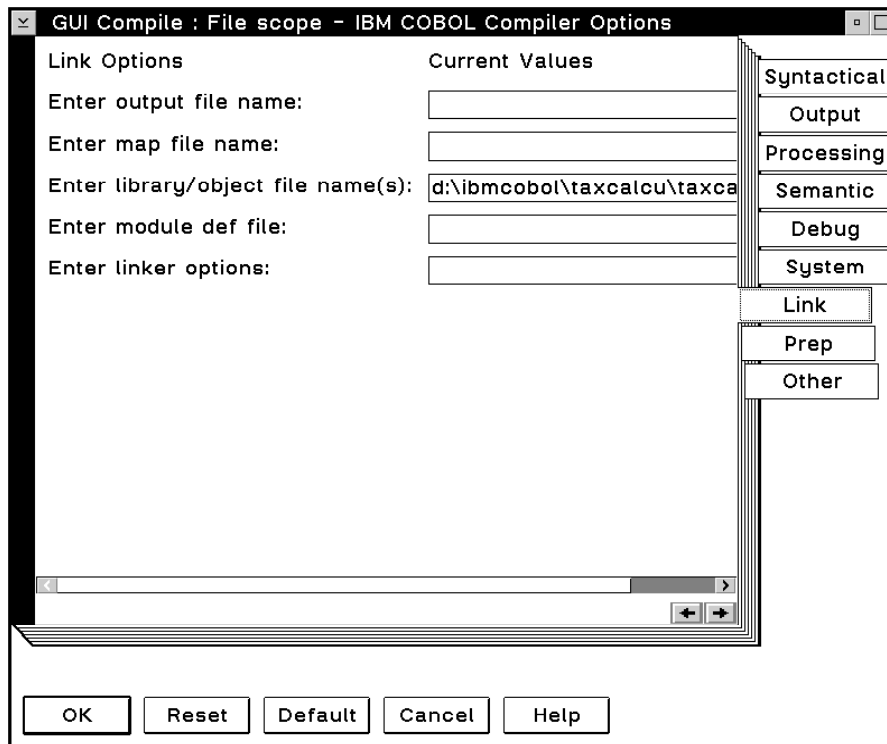


Figure 52. Setting compile options for the Tax Computation application

4. In the **Enter library/object file name(s)** entry field, type:

D:\IBMCOBOL\TAXCALCU\TAXCALCU.OBJ

This causes the Tax Calculation subroutine's object file to be statically linked to the application.

Note: If you specified a different directory when setting up the Tax Calculation subroutine project, specify that directory here.

5. Click on **OK** to close the Compiler Options notebook.

Now you will build the application.

Building the Application

When you build a IBM VisualAge for COBOL for OS/2 GUI application, the source files are compiled and linked to create a running application. If you have nested projects, the inner projects are built first when you build the top project (in this case, the GUI project).

Building the Application

To build your application:

1. From the **Tax Computation Application** window, select the **Project** menu bar choice.
2. Select **BUILD**.
3. When the build starts, the Monitor window appears at the bottom of the **Tax Computation Application - Icon view** window. The Monitor window displays the output of the build. The files that you created for the application are compiled and linked to create the specified target file.
4. When the build completes, the return code is displayed. A return code of zero indicates that your application was built without errors and is ready to run.

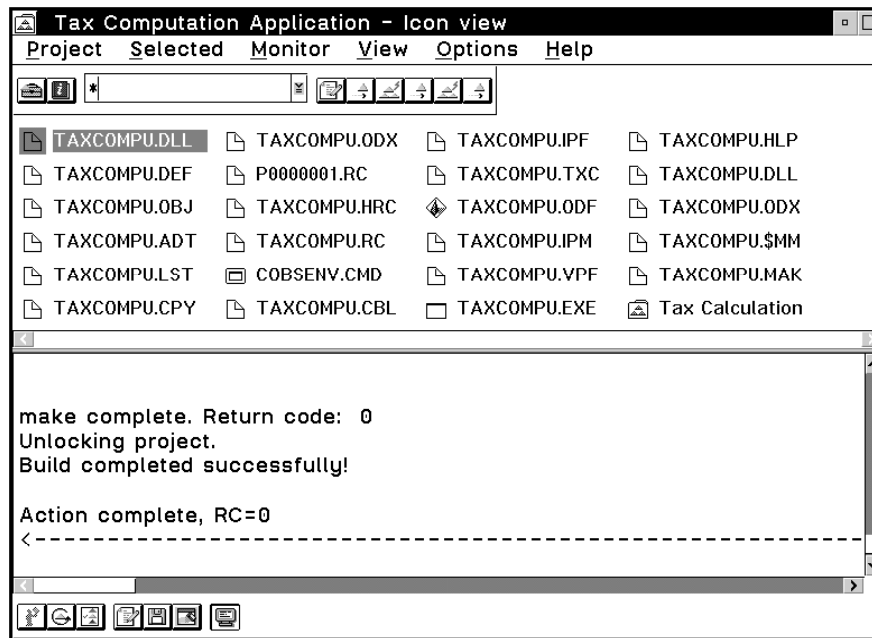


Figure 53. Building the Tax Computation Application

If you do not get a return code of 0, the Monitor window displays error messages. Scroll back up to the error message lines. If you have compile errors, the error message lines will have the drive, path names, and source file name as well as the error message text. Double-click on a compile error message line. The COBOL Editor appears, showing the line in the source file where the error occurred. Correct the error and save the file. To build it again, select **Project**, then select **BUILD**. You will see the results of the second build in the Monitor window.

Debugging Your Application

Using the Interactive Debugger

Use the VisualAge COBOL Interactive Debugger (IDBUG) to debug your program. Using the debugger, you can run your program, set breakpoints, monitor variables, monitor the registers, monitor storage and the call stack.

Preparing Your Application for Debugging

This section discusses how to prepare your application for debug. Before you invoke the interactive debugger, ensure that you followed the instructions in step 3 on page 104 and in step 2 on page 106.

Note: You need to turn off the OS/2 automatic lockup program by opening the Desktop settings (select **Open**, then **Settings**). Select the **Lockup** tab. Ensure that the **No automatic lockup** radio button is selected.

The lockup screen allows keystrokes to pass through the debugger. In the process of unlocking the screen, you may inadvertently give commands to the debugger. This may lead to unpredictable results that require you to reboot your machine.

Debugging Your Application

1. From the **Tax Computation Application - Icon view** window, select **DEBUG** from the **Project** pull-down.
The **Debugger busy. Please wait.** message appears.
2. The **Debug Session Control** and the **Listing** windows open, as shown in Figure 54.

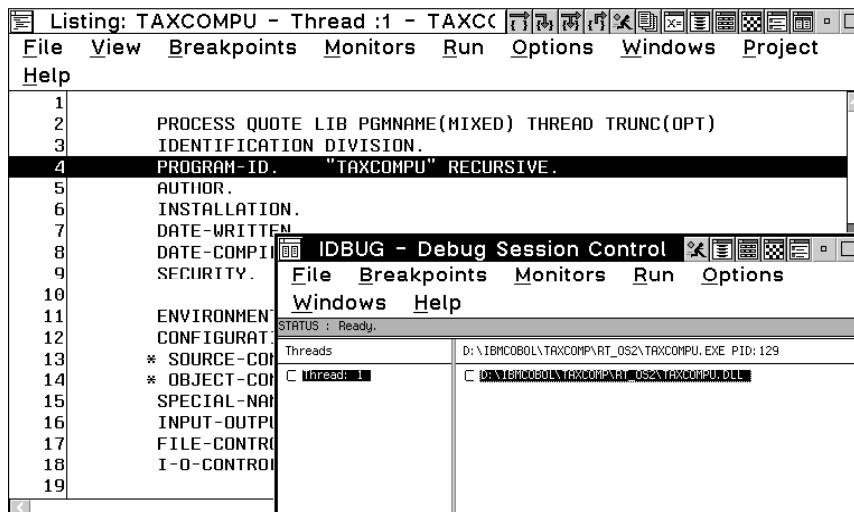


Figure 54. IDBUG - Listing and Debug Session Control Windows

The **Debug Session Control** window is the control window of the debugger and displays during the entire debugging session. One pane of the window shows the

Debugging Your Application

threads for the program you are debugging, the other shows the components of the program.

The **Listing** window shows the source code.

3. On the **Listing** window, select **Options**, select **Debugger settings**, and then select **Debugger properties**. The **Debugger Properties** window appears.

In the **Settings** group box, click on **Set breakpoints at entry points**. This causes the Debugger to stop at all entry points in your application. Click on **OK**.

4. In the **Listing** window, scroll to display the line containing the **getContents** CALL statement. Double-click on the prefix (line number) of this line to set a breakpoint on the CALL statement.

The prefix area is highlighted. You can only set breakpoints on executable statements. The default colors are blue for executable statements and black for non-executable statements.

Note: The debugger is running in synchronous mode. When the debugger is active, the rest of your desktop is suspended.

5. Select **Run** from the **Run** pull-down. Your **Sales Tax Computation** window appears. Type a new value in the **Enter sales amount** entry field. Click on the **Compute tax** push button.

The **Listing** window appears in the foreground. The GUI application stops at the ENTRY statement for the **Compute tax** push button press event. This is a result of turning on the **Set breakpoints at entry points** check box in step 3 on page 111.

6. Select **Run** from the **Run** pull-down. The GUI application stops at the breakpoint set during step 4.

7. To display a variable and change its value:

- a. On the **Listing** window, double-click on the variable you want to display.
- b. A **Program Monitor** window displays the variable and the value currently assigned to it.
- c. To change the value, double-click on the value of the variable in the right-hand column in the **Program Monitor** window, then type over the old value with a new value. Press **Enter**.

8. You can continue to step through and analyze your application by selecting the appropriate options from the **Run** pull-down. Some of those options include the following:

Step over Executes the current line in the program. If the current line is a call, execution stops when the call completes.

Step into Executes the current line in the program. If the current line is a call, execution stops at the first statement in the called program.

Running the Application

Step debug	Executes the current line in the program. The debugger steps over any program for which debugging information is not available (for example, library and system routines), and steps into any program for which debugging information is available.
Step return	Automatically executes the lines of code up to, and including, the return statement of the current program.
Run	Starts and stops the program. When the push button is green in the toolbar, you can start the program. When the push button is red, you can stop the program.

For more information about debugging tasks, select **How do I** from the **Help** pull-down in any of the debugger windows.

Ending the Debugging Session

To end the debugging session, select **Close debugger** from the **File** pull-down in any of the debugger windows. A confirmation message appears. Select **Yes**.

You can also end the debugging session by:

- Pressing **F3** in any of the debugger windows.
- Closing your GUI application. The debugger **Startup Information** window appears. Click **Cancel** to exit or click **OK** to start another debug session.

You are ready to run the Tax Computation application.

Running the Application

From the **Tax Computation Application - Icon view** window, select **RUN** from the **Project** pull-down. The **Sales Tax Computation** window appears.

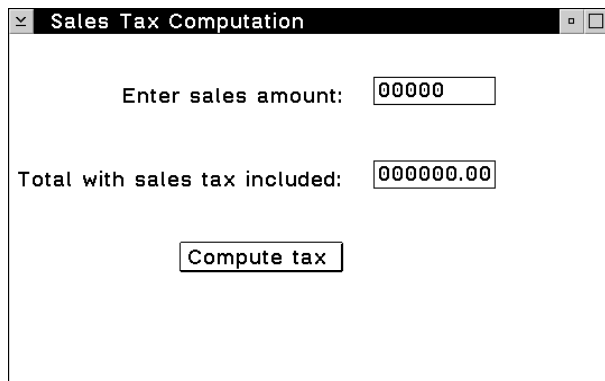


Figure 55. Running the Tax Computation Application

To close the Tax Computation application, double-click on the system menu in the upper left corner of the window.

Packaging the Application for Distribution

Packaging the Application for Distribution

After you have successfully built and run your application, you can package it along with the necessary runtime environment for distribution on diskettes, CD-ROM, or on a local or remote hard disk.

For more information about distributing an application, see the **Task Helper** located in the main **VisualAge COBOL - Icon View** window.

For the Tax Computation application and the runtime environment, you need approximately three or four empty, formatted 2MB diskettes. To package the Tax Computation application:

1. From the **Tax Computation Application - Icon view** window, select **PACKAGE** from the **Project** pull-down. The **Package VisualAge COBOL Application** appears.

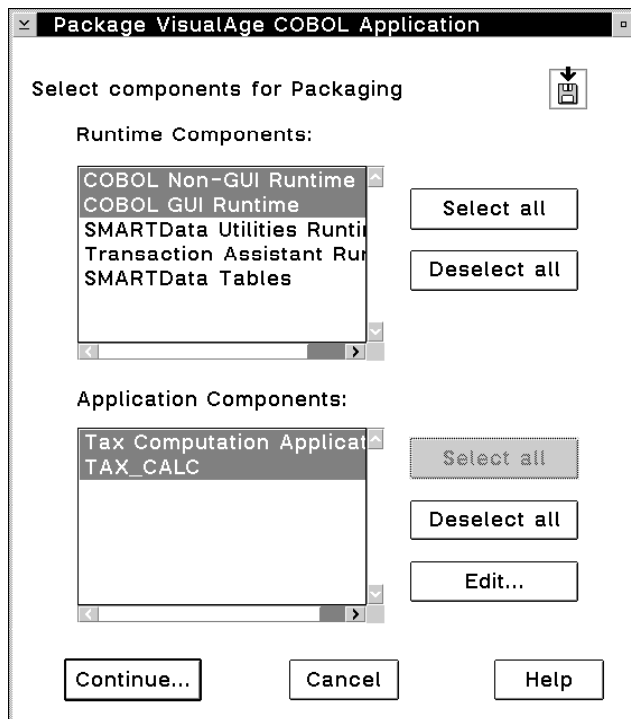


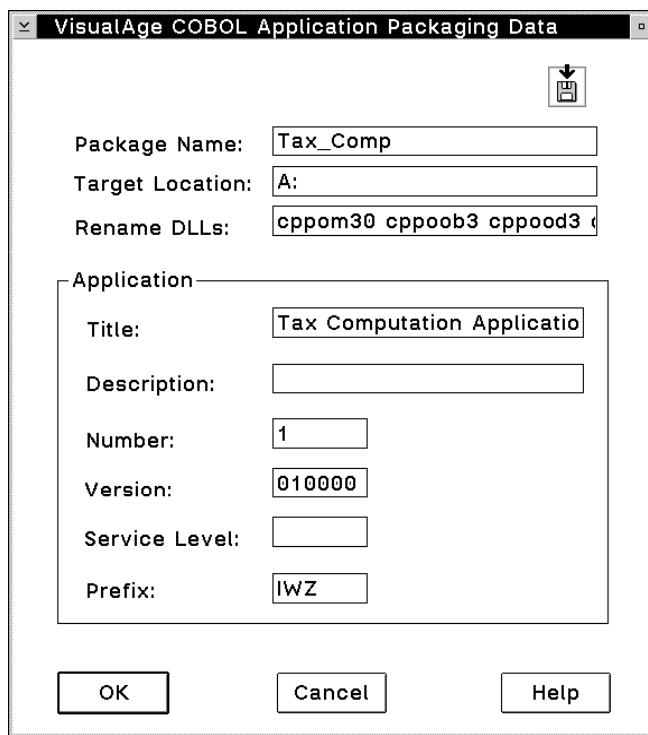
Figure 56. Package VisualAge COBOL Application window

2. Ensure that the **COBOL Non-GUI Runtime** and **COBOL GUI Runtime** components are selected in the **Runtime Components** list box.
3. Ensure that the **Tax Computation Application** is selected in the **Application Components** list box.

Packaging the Application for Distribution

Note: Depending on the file system, the name of your project may be all upper case or in mixed case.

4. Click on **Continue**.
5. The **VisualAge COBOL Application Packaging Data** window appears.



The screenshot shows a dialog box titled "VisualAge COBOL Application Packaging Data". It contains several input fields and a sub-dialog box. The main fields are: "Package Name" with the value "Tax_Comp", "Target Location" with the value "A:", and "Rename DLLs" with the value "cppom30 cppoob3 cppood3". The sub-dialog box, titled "Application", contains: "Title" with "Tax Computation Applicatio", "Description" (empty), "Number" with "1", "Version" with "010000", "Service Level" (empty), and "Prefix" with "IWZ". At the bottom are "OK", "Cancel", and "Help" buttons.

Figure 57. VisualAge COBOL Application Packaging Data window

6. Accept the defaults in the entry fields. Subsequent steps assume that your **Target Location** is A:. Your Tax Computation application will be packaged onto diskettes in drive A:.
7. Insert an empty, formatted diskette (label it Disk 1) into drive A: and click **OK**.
8. A message informs you that the first phase of the packaging process is about to start. Click **OK** on the message dialog.
9. The **Application Packaging Status** window appears and starts the first phase of packaging. This may take a few minutes.
10. After the first phase of packaging is complete, a message appears informs you that the first phase is complete. Click **OK** in the message dialog.
11. The **Diskette Generator** window appears.

Installing on an End User Machine

A message in the Diskette Generator window informs you that you should insert the first distribution diskette into drive A:. Ensure that the first diskette is still in drive A:. Press **Enter**.

12. The Diskette Generator window prompts you for more diskettes when necessary.
13. After the Diskette Generator has completed, a message is displayed that prompts you to reinsert the first diskette. Reinsert the first diskette and click **OK**.
14. When the packaging process is complete, a message informs you that the packaging process has completed successfully. Click on **OK**.

Installing the Application on an End User Machine

After you have packaged your application, you can distribute the diskettes to your users. Install your application from the diskettes as follows:

1. Insert the first diskette into drive A: of a user machine.
2. Open an OS/2 window and type: a:\install. Press Enter.
3. The **Installation** window appears.
4. Read the instructions in the **Instructions** window and click **Continue**.
5. The **Install** window appears.
6. Accept the defaults and click **OK**.
7. The **Install - directories** window appears.
8. Click **Select all** to select all components to install. Accept the default installation directories as shown or modify them. Click **Install**.
9. The **Install - progress** window appears.
10. The Installation program prompts you to insert diskettes when necessary.
11. When the installation process is complete, a message informs you that all components are successfully installed.
12. Remove any diskettes from drive A:. Shut down and reboot the user machine to activate the changes in the CONFIG.SYS file.
13. After rebooting, open the **Tax Computation Application** folder on the desktop.
14. To run the Tax Computation Application, double-click on the **TAXCOMPU** icon. If the **Sales Tax Computation** window is not in full view, press **Alt+F7** and use your mouse to move and display the entire window.

Installing on an End User Machine

Examining Tables in the Database Schema View

Creating SQL Statements with Data Assistant

Data Assistant simplifies the process of constructing syntactically correct, embedded SQL statements. It gives you a graphical view of your relational database, allows you to map COBOL data structures to the database, and generates SQL statements into your source file.

Data Assistant is made up of three views. In this brief tutorial, you will use the following steps to open these views and create an embedded SQL statement that lists all the managers in the Employee table of the SAMPLE database:

1. Examine tables of a relational data base (SAMPLE) in the **Database Schema** view.
2. Copy a selected table to the **Data Structure Mapping** view.
3. Map a data structure to selected columns of the table in the **Data Structure Mapping** view.
4. Use this data structure mapping in the **SQL Construction view** to build an SQL statement and insert it into your COBOL code.

While you are going through this tutorial, you may want more information about Data Assistant. You can access online information by pressing the **F1** key.

Note: Data Assistant requires DB2 for OS/2 Version 2.1. This tutorial assumes you have installed the SAMPLE database. It also assumes that you have already created a project, either by using the instructions in "Build Your First VisualAge COBOL Application" on page 43 or by creating a project of your own.

Examining Tables in the Database Schema View

First, you will examine the tables of the SAMPLE database in the Database Schema view. You will select one of these tables in the data structure mapping in the next step.

To open the Database Schema view and examine the tables in the SAMPLE database:

1. Open a project. If you have not created one, see the instructions in "Build Your First VisualAge COBOL Application" on page 43.
2. Start **DB2 for OS/2** before you proceed to open the Database Schema view.
 - a. Select the **Project** menu-bar choice, then select the arrow button next to the **Data Tools** choice.
 - b. A cascaded menu appears. From the cascaded menu, select **DB2 Start**.
 - c. When prompted, type the appropriate user ID and password. Click on **OK**.
3. Click on **Project**, then the arrow to the right of **Data Tools**, and then **Data Assistant Schema view**. A prompt appears for you to enter the name of the database. Type SAMPLE and click on **OK**. The Database Schema view opens, displaying the tables of the SAMPLE database. The Schema view displays views

Copying Tables

using a slightly different icon. However, the SAMPLE database contains only tables.

Note: There are eight tables in the SAMPLE database. You may need to scroll or enlarge the window to see them all. This is a read-only view of the database. Data Assistant does not modify the actual database.

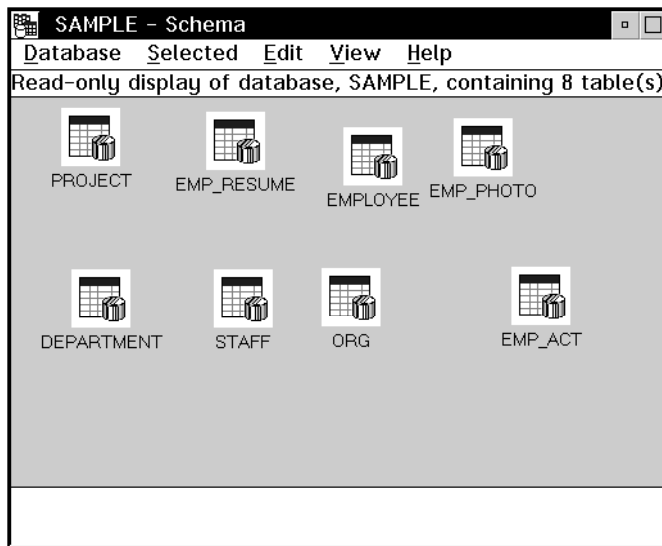


Figure 58. Database Schema View

4. Double-click on any table with mouse button 1 to expand it and see the columns that make up the table. Minimize the table by double-clicking on the title bar of the table.

Note: Do not close the Database Schema view.

In the next steps, you will open the Data Structure Mapping view and copy a table over from this Database Schema view.

Copying Tables to the Data Structure Mapping view

To open the Data Structure Mapping view:

1. From your project, select the **Project** menu-bar choice, then select the arrow button next to the **Data Tools** choice.
2. A cascaded menu appears. From the cascaded menu, select **Data Assistant Mapping view** from the **Data Tools** menu choice. A prompt appears for you to enter the mapping file name.
3. Type DA-TUT and click on **OK**. The Data Structure Mapping view opens.

Copying Tables

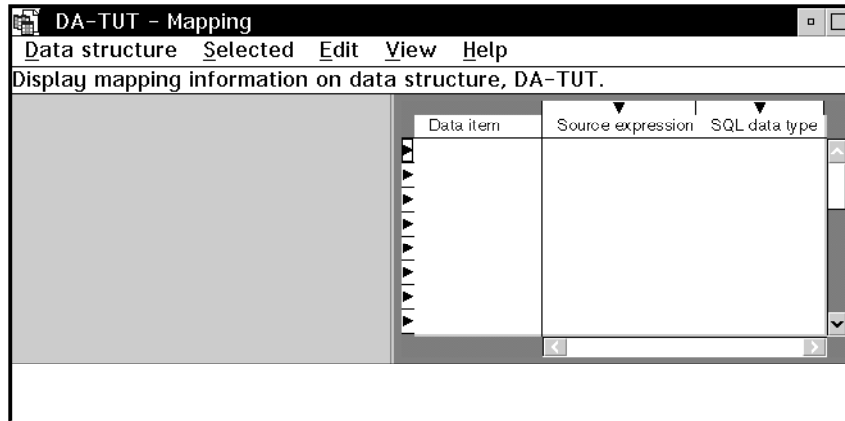


Figure 59. Data Structure Mapping View

The Mapping view is made up of two panes. The left pane will contain tables and views you copy over from the Database Schema view. The right pane is where you specify data items, which typically map to columns of the tables and views in the mapping view.

To create the data structure mappings that are used to create SQL statements, you need to copy the tables and views that your application needs from the Database Schema view over to the Data Structure Mapping view. In this tutorial, you will use the **Staff** table.

To copy the **Staff** table over to the Mapping view:

1. Locate the **Staff** table in the Database Schema view.
2. Press and hold mouse button 2 on the **Staff** table icon while dragging the object from the Database Schema view window over to the Mapping view window, then release mouse button 2.

The table icon will be copied into the Mapping view window.

Mapping Data Structures

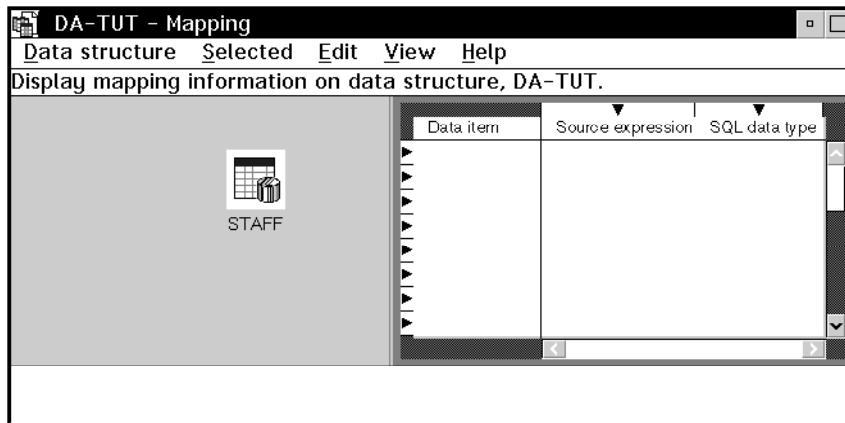


Figure 60. Data Structure Mapping View with Staff Object

3. Close the Database Schema view (not the Mapping view) by double-clicking on the upper left corner of the title bar.

Now you are ready to create a data structure that maps the data items to the database.

Mapping Data Structures

In the following steps, you will map a data structure to the SAMPLE database. Later, you will use the data structure in the SQL Construction view to create your SQL statement.

1. Expand the **Staff** table by double-clicking on it. Now you can see the columns that make up the **Staff** table.
2. Click on the **Name** column of the **Staff** table. The data item name, source expression value, and SQL data type are entered into the data structure mapping.
3. Select the **Dept**, **Job**, and **Salary** columns in the same manner.

If you want to delete a data item from the mapping, click on the selection indicator (the arrows that run down the left side of the mapping table) next to the item you want to delete and select **Delete Data Item** from the **Selected** menu.

Mapping Data Structures

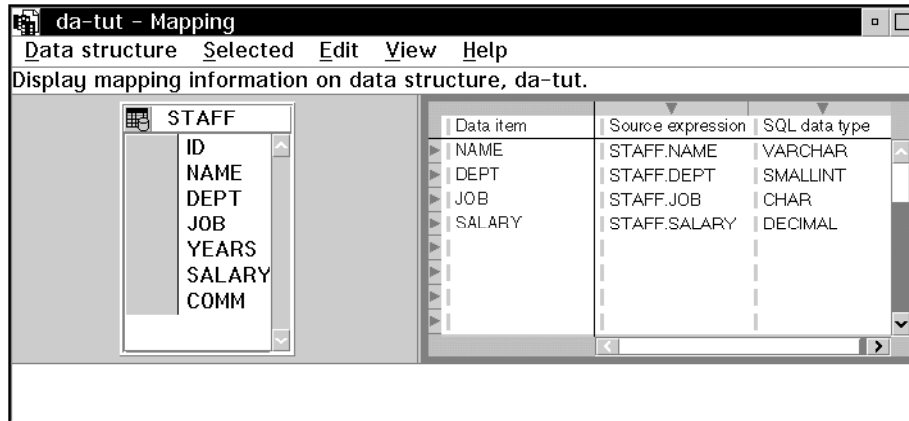


Figure 61. Completed Data Structure Map

- To save your data structure, select **Save** from the **Data Structure** menu. A message box appears telling you that three files have been saved:

- DA-TUT.INI (an internal INI file)
- DA-TUT.SM (an internal mapping file)
- DA-TUT.CPY (a copybook file)

Click on **OK**.

Note: These files are saved in a subdirectory off of the directory specified by the ICODDIR environment variable in your CONFIG.SYS file. This subdirectory is named `\DA`. For example, if you type `SET ICODDIR` on your OS/2 command line and the response is `C:\COBOL`, the data structure mapping files are saved in the directory `C:\COBOL\DA`.

- Select **Report** from the **Data Structure** menu bar to generate a report about the contents of the data structure mapping. A message appears to tell you that the report has been saved as the text file `DA-TUT.RPT` in the `\DA` subdirectory of the directory specified by the ICODDIR environment variable. You can view this report using any text editor.
- Click on **OK** to close the message window.
- Close the Data Structure Mapping view by double-clicking in the upper left corner of the title bar. A message appears asking if you want to save the mappings. Click on **Exit**.

Note: To see other ways of adding or updating data items in the data structure mapping, select **How Do I...** from the **Help** menu bar, then select the appropriate topic from the list.

With this simple data structure mapping, you are now ready to create your SQL statement.

Once you have built a catalog of reusable data structure mappings, you will be able to create most of the SQL statements you need without going back to the Database

Creating an SQL Statement

Schema or Data Structure Mapping views. You can then easily access the data structures from the SQL Construction view. In many cases, a database administrator or team leader is likely to create the necessary mappings. Then the application developer begins programming as described in the next step.

Creating an SQL Statement

In these steps, you will open the SQL Construction view and use the DA-TUT.SM data structure you just mapped to build an SQL statement.

To open the **SQL Construction** view:

1. From your project, select the **Project** menu-bar choice, then select the arrow button next to the **Create** choice.
2. A cascaded menu appears. From the cascaded menu, select **Create New Text File with the COBOL Editor**. The New window appears, in which you specify options for your new file.
3. In the New 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. This sets the COBOL language-sensitive editing features on.
4. Click on the **New** pushbutton. The COBOL Editor displays the window titled Editor - Untitled Document 1 and recognizes the file as a COBOL file.

You can tell that the language-sensitive editing features are on by checking to see that the format line, the line just below the menu bar, displays *, A, and B. If you cannot see the format line, you can display it by selecting **View**, then selecting **Format line**.

Note: For the Data Assistant toolbar icon and menu choices to appear, you must be editing a COBOL file with the extension **.CBL**, a copyfile with the extension **.CPY**, a COBOL DB2 file with the extension **.SQB**, or a CICS file with the extension **.CCP**.

5. Select **Insert Code** from the **Edit** menu, then select **SQL** from the **Insert Code** menu choice. This brings up the Data Assistant **SQL Construction** view.

The following steps will create a simple SQL **SELECT** statement that lists the departments, positions, names, and salaries of all managers listed in the **Staff** table.

1. Click on the drop-down list button to the right of the **SQL statement type** list box to see the list of SQL statements supported. Select **SELECT** from the displayed list.
2. Click on DA-TUT.SM in the **Available data structure(s)** window to select the data structure you mapped earlier. The **Available data item(s)** list box displays the columns you selected in the Data Structure Mapping view.
3. Click on DEPT, JOB, NAME, and SALARY from the **Available data item(s)** window. Notice that an SQL statement is created in the **SQL statement appear-**

Creating an SQL Statement

ance window as you select the data items. You will need to scroll up to see the entire statement.

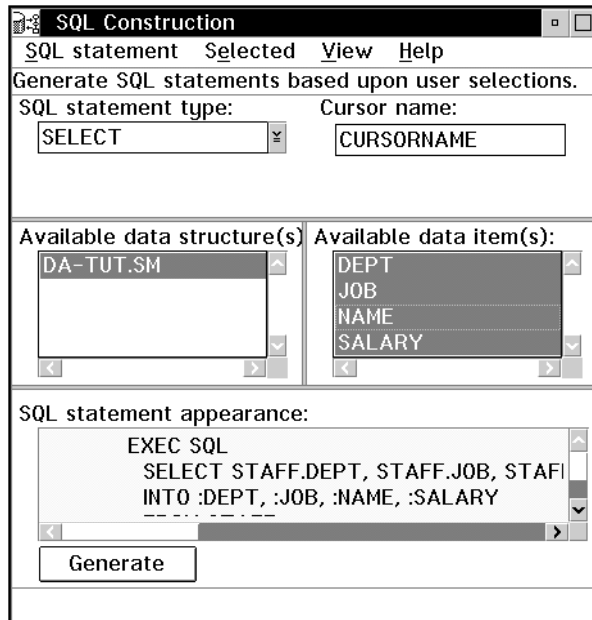


Figure 62. Data Assistant SQL Construction View

In this next step, you will add a condition to the SQL statement so that it selects only those staff members listed as managers.

1. Select **Condition** from the **SQL statement** menu bar. This opens the **Condition** action window.
2. Click on **JOB** from the **Operands & operators** window. **STAFF.JOB** appears in the **Condition (in SQL Syntax)** window.
3. Click on **Comparison operators** from the **Operand & operator types** window.
4. Click on **Equal** in the **Operands & operators** window.
5. Click in the **Condition** window and type a space, then 'Mgr' after the equal (=) sign.

Creating an SQL Statement

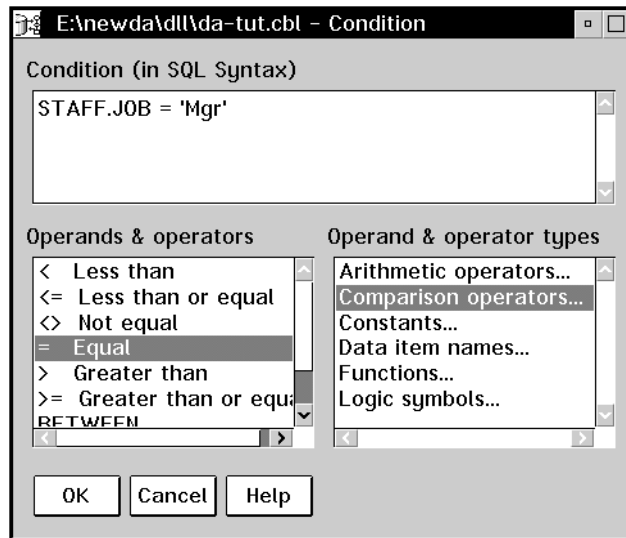


Figure 63. Condition Action Window

6. Click on the **OK** push button. The **Condition** action window disappears and the condition is entered in the **SQL Statement Appearance** window. Scroll up to view the condition.
7. In the SQL statement appearance area, scroll up to see the entire SELECT statement.
8. Click on the **Generate** push button to insert the code into the COBOL Editor session at the current cursor position.
9. Click in the COBOL Editor to see the tokens colored.

Creating an SQL Statement

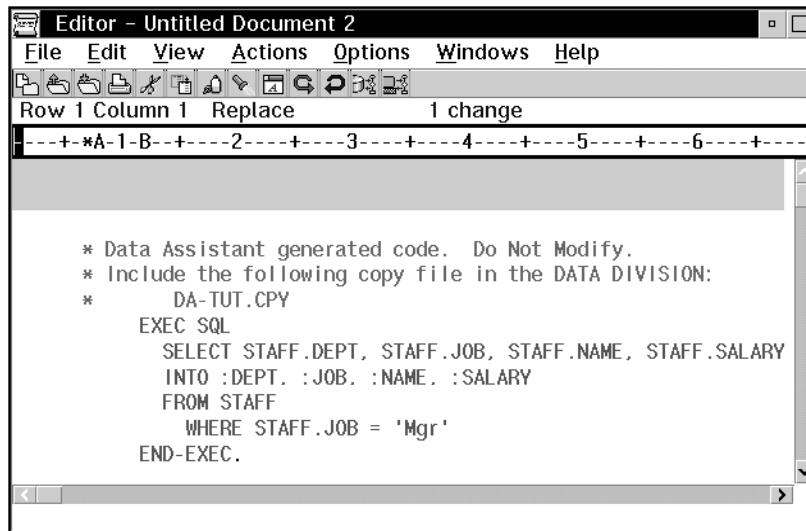


Figure 64. SQL Statement Inserted Into COBOL Editor

You have now successfully created an embedded SQL statement. As noted in the generated comments, remember to include the generated copy file in the Data Division of your program.

10. Double-click on the upper left corner of the SQL Construction view to close the view.
11. Double-click on the upper left corner of the COBOL Editor session to close the session. A window appears asking if you want to save the file. It is not necessary to save this file because it is not a complete COBOL program. Click on the **No** push button. The file closes.

As noted previously, the ability to save and reuse data structure mappings makes Data Assistant more efficient each time you use it. Though you went through the entire process of mapping a data structure to a database and then generating an SQL statement, you will be able in later sessions to go straight to the SQL Construction view and reuse the same mapping to create new statements.

Creating an SQL Statement

Using the CICS Transaction Assistant

Using the CICS Transaction Assistant

Transaction Assistant simplifies the task of constructing CICS transaction calls in COBOL programs. The Assistant takes the information you enter into a dialog window and uses it to generate a CICS ECI call and parameter list for invoking CICS transactions.

In this tutorial, you will fill in the required information and generate a CICS call. To use the Transaction Assistant, you must be editing in the COBOL Editor.

If you do not have a COBOL Editor session open, do the following:

1. From your project, select the **Project** menu-bar choice, then select the arrow button next to the **Create** choice.
2. A cascaded menu appears. From the cascaded menu, select **Create New Text File with the COBOL Editor**. The New window appears, in which you specify options for your new file.
3. In the New 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. This sets the COBOL language-sensitive editing features on.
4. Click on the **New** pushbutton. The COBOL Editor displays the window titled Editor - Untitled Document 1 and recognizes the file as a COBOL file.

You can tell that the language-sensitive editing features are on by checking to see that the format line, the line just below the menu bar, displays *, A, and B. If you cannot see the format line, you can display it by selecting **View**, then selecting **Format line**.

Or, if the COBOL Editor is already open, start a new file named tatest.cb1.

Note: Transaction Assistant is only available in the COBOL Editor when you are editing the following files:

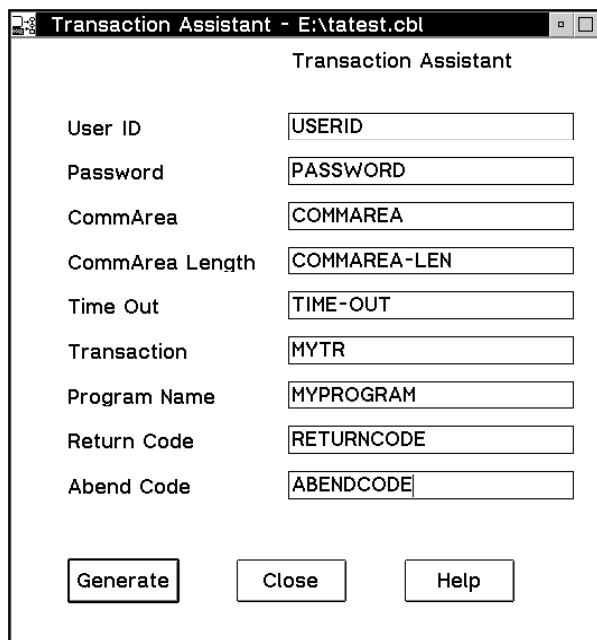
- COBOL (.cbl)
- Copy (.cpy)
- DB2 (.sqb)
- CICS (.ccp)

To open the Transaction Assistant:

1. Select **Insert Code** from the **Edit** menu.
2. Select **CICS ECI** from the **Insert Code** menu choice.

The **CICS transaction assistant** window opens. Figure 65 on page 128 shows this window completed with sample information.

Using the CICS Transaction Assistant



Transaction Assistant	
User ID	USERID
Password	PASSWORD
CommArea	COMMAREA
CommArea Length	COMMAREA-LEN
Time Out	TIME-OUT
Transaction	MYTR
Program Name	MYPROGRAM
Return Code	RETURNCODE
Abend Code	ABENDCODE

Generate Close Help

Figure 65. CICS Transaction Assistant Window

3. To see how the Transaction Assistant generates a CICS transaction call and inserts it into your COBOL code, fill in the entry fields as shown in the example, then click on the **Generate** push button to insert the code at the current cursor location in your COBOL file.

Or, if you want to generate a working CICS transaction call, use the COBOL file of your choice and the descriptions of each field below to fill in the following entry fields based on information at your site.

Note that while CICS may place limitations on the size of the value a program variable represents, Transaction Assistant does not limit the size of the program variable names.

Also note that Transaction Assistant does not examine or validate the program variables you use in your program.

For more information about programming considerations or the information required for these fields, click on the **Help** push button, or place the cursor in any field and press **F1**.

User ID Enter the name of a program variable containing the CICS USERID that will be used to make the call. CICS limits the length of the actual User ID to eight characters.

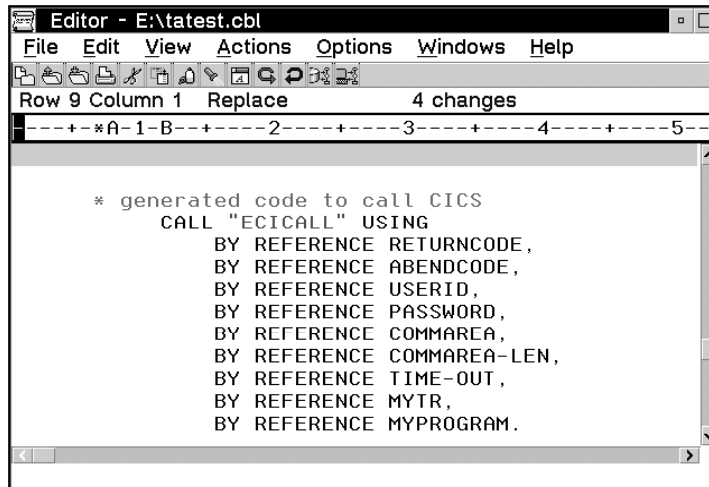
Define the USERID in the Data Division as PIC X(8).

Using the CICS Transaction Assistant

Password	<p>Enter the name of the variable defined for the password associated with the USER ID. CICS limits the length of the password to eight characters.</p> <p>Define the password in the Data Division as PIC X(8).</p>
CommArea	<p>The <i>commarea</i> is a buffer used to pass information between CICS and the COBOL program.</p> <p>If a commarea is not needed for this call, leave the CommArea field blank.</p> <p>If a commarea is needed, enter the name of the commarea in the CommArea entry field, and enter the length of the commarea, in bytes, into the CommArea Length entry field. CICS limits the length to 16K bytes.</p> <p>The Data Division definition of this field is determined by the application programmer.</p>
CommArea Length	<p>The <i>commarea length</i> is the length of the commarea buffer in bytes.</p> <p>If a commarea is not needed for this call, leave the CommArea Length entry field blank.</p> <p>If a commarea is needed, enter the name of the commarea in the CommArea entry field, and enter the length of the commarea, in bytes, into the CommArea Length entry field. CICS limits the length to 16K bytes.</p> <p>Define The commarea length in the Data Division as PIC S9(9) COMP-5.</p>
Time Out	<p>Enter the name of a program variable containing the length of time, in seconds, the application will wait for the CICS program to complete. You can also enter a literal value into this field.</p> <p>Define the timeout in the Data Division as PIC S9(9) COMP-5.</p>
Transaction	<p>Enter the program variable defined in your COBOL program for the CICS transaction code under which the program will run. This field is optional and can be left blank.</p> <p>If the transaction code is not specified, the program will run with default properties. If it is provided for you, define it in the Data Division as PIC X(4).</p>
Program Name	<p>Enter the name of a program variable defined in your COBOL code for the name of the CICS server program being called. CICS limits the length of the name of the program to 8 characters.</p>

Using the CICS Transaction Assistant

- Define the program variable in the Data Division as PIC X(8).
- Return Code** The return code is the name of the COBOL program variable that will be set by CICS when the application is run. The name of the variable must be 32 characters or less.
- A negative return code indicates that the ECI errors are defined by CICS OS/2.
- A zero or positive return code indicates that the ECI errors are from the called CICS program.
- Define the return code in the Data Division as PIC S9(9) COMP-5.
- Abend Code** The abend code is the name of the COBOL program variable that will be set to the value returned by CICS if the transaction abnormally terminates.
- Define the abend code in the Data Division as PIC X(4).
4. When you have completed the fields in this window, click on the **Generate** push button.
 5. The code for making the call is placed in the editor window at the current cursor position. Figure 66 shows the code generated for the parameters entered in Figure 65 on page 128.
 6. Click on the **Close** push button to close the Transaction Assistant.



```
Editor - E:\tatest.cbl
File Edit View Actions Options Windows Help
Row 9 Column 1 Replace 4 changes
-----*A-1-B-----2-----3-----4-----5-----
* generated code to call CICS
CALL "ECICALL" USING
  BY REFERENCE RETURNCODE,
  BY REFERENCE ABENDCODE,
  BY REFERENCE USERID,
  BY REFERENCE PASSWORD,
  BY REFERENCE COMMAREA,
  BY REFERENCE COMMAREA-LEN,
  BY REFERENCE TIME-OUT,
  BY REFERENCE MYTR,
  BY REFERENCE MYPROGRAM.
```

Figure 66. Editor Window with Code Generated by CICS Transaction Assistant

Appendix A. Using OS/2

This appendix describes how to perform basic tasks in OS/2: using your mouse, using windows, and using controls in windows.

Using the Mouse

Pointing devices, such as a mouse, are the key to any graphical user interface. The mouse allows you to interact with the objects on your desktop, the VisualAge COBOL product, and the OS/2 operating system.

The mouse at your workstation typically has two buttons on it. Each button performs different operations on the OS/2 desktop. The configuration of button operations can be tailored to your liking.

The following list describes the default operations for each mouse button.

Button	Function
1	Button 1 (usually the left button) is used for selecting objects, opening files, and starting applications.
2	Button 2 (usually the right button) is used for moving or manipulating objects.

The following list provides definitions of mouse manipulations.

Manipulation	Function
Click	To press and release a mouse button.
Double-Click	To press and release a mouse button twice in rapid succession.
Drag	To press and hold a mouse button while moving the mouse. Dragging ends when the mouse button is released. For example, dragging causes an icon to be moved to another location on the screen.
Drag-and-Drop	To directly manipulate an object by dragging it onto another object. This transfers information from a source object to a target object. For example, dragging a document object to a printer object causes the file associated with the document to be sent to the printer associated with the printer object. Button 2 is usually used during a drag-and-drop.

Use your mouse on a flat, smooth surface such as your desk, or a mouse pad. Move your mouse across this surface. If you run out of desk space while moving your mouse, simply pick up the mouse and place it in a more comfortable location before moving it again.

The movement of the mouse across a surface rolls a ball inside of the mouse which, in turn, sends information to the computer. This information allows the mouse pointer on the desktop to follow the mouse movement.

Using OS/2

The mouse pointer usually looks like an arrow, but it can take on a different appearance, depending on the software you are using, the actions that you are performing, and the availability of computer resources. For instance, it may look like a clock. This is the wait pointer, which indicates that the current application is busy and that no additional action can be performed on it.

Working with Windows

A window is an area of the screen with visible boundaries in which you can view information or interact with an application.

The parts of a window enable you to work with the window. Window parts you will be using are labelled in the diagram below.

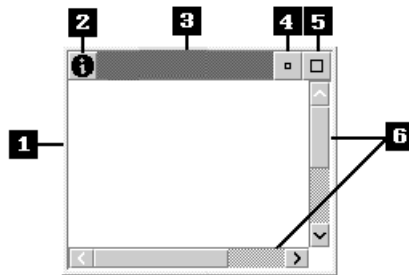


Figure 67. Parts of a Window

The parts of the window are as follows:

1. Border

Indicates the boundaries of a window.

2. System-menu symbol

Gives the user access to choices that affect the window.

3. Title bar

Contains the system-menu symbol, the window title (the object and view name), and the window sizing buttons.

4. Minimize button

Reduces the window to its smallest possible size.

5. Maximize button

Enlarges the window to its largest possible size.

6. Scroll bars

Indicate that there is more information in the window and contain controls to access the information.

Invoking a Window: In OS/2, icons on the desktop are used to start a process, such as the VisualAge COBOL product, or to open an OS/2 window.

Double-clicking on an icon is one way to open a window or start a process in OS/2.

Using OS/2

You can also click on an icon with mouse button 2 to display a pop-up menu for the icon. One of the choices is usually **Open**.

Sizing a Window: You can reduce the size of a window to create more space on the OS/2 desktop or enlarge it to see more information in the window.

To change the length or width of a window:

1. Move the mouse pointer to any point in the border of the window until the mouse pointer changes to a double-headed arrow.
2. When you see the double-headed arrow, press and hold the right mouse button while dragging the mouse up or down from the window, then release the mouse button.

The window becomes longer as you drag the border down or shorter as you drag the border up. You can use any of the four borders to resize a window.

3. You can also change the length and width of the window at the same time.

Move the mouse pointer to the corner of the border on the window until the mouse pointer changes to a diagonal double-headed arrow.

4. When you see the diagonal double-headed arrow, press and hold the right mouse button while dragging the mouse toward the inside or outside of the window, then release the mouse button.

The length and width of the window becomes smaller or larger as you drag the corner of the border inward or outward.

Moving a Window: Many times you will need to access windows or icons that are located behind a window.

One way to access those objects is to move the window to reveal the objects behind it. Moving windows is also a way to arrange the windows on your desktop to suit your needs.

To move a window, position the mouse pointer on the title bar at the top of the window. Press and hold mouse button 2 while dragging the mouse, then release the mouse button.

The window moves in the direction you move the mouse. When you release the mouse button, the window is relocated on the desktop.

Minimizing a Window: When you have many windows open on your desktop, you may want to remove one or more of them from view.

Minimizing a window reduces it to an icon and removes the window from view. When you minimize a window, you can quickly bring it back to view as it was before you minimized it. You can locate the window using the OS/2 window list, as described in "Locating Windows" on page 134.

Using OS/2

To minimize a window, click on the minimize button, the button next to the rightmost button in the upper right-hand corner of the window.

The window reduces to an icon. The icon may be hidden from view. It is located either on the OS/2 desktop or in the Minimized Window Viewer.

Locating Windows: When you have minimized a window and need to locate it again, you can use the OS/2 Window List to make it visible again.

To access the OS/2 Window List press the **Ctrl+Esc** key combination. To do this, press and hold the **Ctrl** key, then press and release the **Esc** key.

The OS/2 Window List appears. It displays a list of the open windows, including open windows that are minimized.

Note: The OS/2 Window List closes automatically if you click the mouse button outside of the list. If this happens, press **Ctrl+Esc** again.

To bring a window back into view, locate its title in the OS/2 Window List. You may need to scroll to find it. When you have found its title, double-click on it. The window reappears on the desktop as it looked before it was minimized.

Maximizing a Window: Maximizing a window enlarges it to its largest possible size. Restoring a window returns it to the size it was before you maximized it.

When you maximize a window, it enlarges to the largest size possible for the particular view or to the size of the workplace. The maximized window often covers the entire desktop. To maximize a window, click on the maximize button, the rightmost button in the upper right-hand corner of the title bar in the window.

After you maximize the window, the rightmost button in the upper right-hand corner of the title bar is replaced with the restore button. Click on the restore button to change the window back to the size it was before you maximized it.

Activating a Window: The active window is the window with which you are currently interacting. This is the window that has a highlighted border and title bar. When working with multiple windows, it is important to be aware of which window is active.

There are several ways to make a window active.

- If a window is visible, you can make it active by clicking on it.
- When you open a window, it becomes the active window.
- You can use the OS/2 Window List to make a window active.

Any keyboard input is applied to the active window. When using your keyboard to interact with windows, you must be aware of which window is active. For example, if you try to type in an editor window when it is not active, your text will not appear there. OS/2 will try to apply what you type to whatever window is active at the time. Unintended results may happen if you have not made the appropriate window active before performing an action.

Using OS/2

Closing Windows: When you are finished using a window, you can close the window.

There are several ways to close a window.

- You can click once on the system-menu symbol in the upper left-hand corner of the window. The system menu appears, and **Close** is one of the choices. Click once on **Close** to close the window.
Note: The system menu closes automatically if you click the mouse button outside of the menu. If this happens, click once on the system-menu symbol again.
- You can double-click on the system-menu symbol of the window. In general, this action closes the window, as close is the default menu choice.

Unlike minimized windows, you cannot access closed windows from the OS/2 Window List. You must invoke the window again to use it.

Using Controls on a Window

This topic describes how to use some of the controls that can be found on a workstation window. You will encounter these controls when using VisualAge COBOL.

Using Scroll Bars in Windows: Most windows have more information in them than can be displayed in the window at one time. The long, vertical grey area at the right edge of a window or at the bottom of a window is the scroll bar.

These vertical and horizontal scroll bars allow you to move unseen information into view. Using a mouse, there are three ways to scroll through a window:

- When there is off-screen information, there will be a light-colored part (the scroll box) on the scroll bar.
 - Click on the scroll box in the side scroll bar and, while holding the button, drag it along in either direction. This moves the information up or down in the window.
 - Click on the scroll box in the bottom scroll bar and, while holding the button, drag it along in either direction. This moves the information right or left in the window.
- There are arrows at each end of the scroll bar. When there is off-screen information, the arrow will not be grayed out.
 - Click on a non-grayed arrow on the side scroll bar to move the information up or down in the window.
 - Click on a non-grayed arrow on the bottom scroll bar to move the information right or left in the window.
- When there is a light-colored part (the scroll box) on the scroll bar, click on the dark area above or below the scroll box. Information scrolls by an entire window-full.

Using OS/2

Using Menus in Windows: One way of performing actions in a window is to use its menus.

There are different types of menus you might find available in OS/2 windows:

- Menu available from the menu bar at the top of the window.
To access these menus, click on the menu-bar choice. This is located below the title bar on a window. Select a choice by clicking on it.
- Pop-up menus that you can access within a window using the right mouse button.
To access these menus, move your mouse pointer to a blank part of the window or to an object in the window (such as an icon) and click with the right mouse button. Select a choice by clicking on it.

Using Push Buttons: Push buttons are used in windows for actions that occur immediately. However, if you see a push button with an ellipsis (...) after the text within it, clicking on the push button will display another window.

To perform an action with a push button, move the mouse pointer over the button and click with mouse button 1.

Using Radio Buttons and Check Boxes: Radio buttons and check boxes are controls you may find on windows. They enable you to make a choice from a set of options.

Radio buttons are used to select only one item from a number of choices. When you click on one radio button in a group of radio buttons, that button is filled in with a dot. This means that only this choice is selected. You can change your selection by clicking on another radio button in the group, but only one radio button in a group can be selected at a time.

Check boxes are used to select one or more items from a number of choices. When you click on a check box, that box is filled in with a check mark. This means that the choice is selected. You can make more selections by clicking on other check boxes in the group, and all of your selected choices will apply. You can deselect a check box by clicking on the check box again.

Using Entry and Output Fields: Entry fields are areas in a window in which you enter text. Output fields are areas in a window in which text is displayed only.

To enter text in an entry field, move the mouse pointer over the entry field. You will know when your mouse pointer is over an entry field, because it will change to an I-beam pointer. Click once in the entry field. The text cursor, a blinking vertical bar, appears in the entry field. This means that the text you type will appear in the entry field.

Using OS/2

Using List Boxes: A list box contains a scrollable list of choices that you select. There are several variations of list boxes that you may encounter:

- List box
To use a list box, scroll through the list and click on a choice to select it. It is the selected choice until you click on another choice.
- Drop-down list box
A drop-down list box is a list box that shows one item and a drop-down list button that, when clicked on, displays the full list of choices. Click on a choice to select it.
- Combination box
A combination box has an entry field and a list box with selectable choices. You can either click on a choice in the list box or type a choice in the entry field.
- Drop-down combination box
A drop-down combination has an entry field and a list box with selectable choices. The list box is hidden until you display it. You can either click on a choice in the list box or type a choice in the entry field.

Using OS/2

VisualAge COBOL Sample Applications

Appendix B. VisualAge COBOL Supplied Sample Applications

VisualAge COBOL provides two sets of sample applications: “Employee Lookup Application Samples” and the “SMARTdata Utilities Samples.”

The sample applications are installed when their corresponding components of VisualAge COBOL are installed. The Employee Lookup Application samples are installed when you install the GUI Designer component. The SMARTdata UTILITIES samples are installed when you install the SMARTdata UTILITIES component.

Employee Lookup Application Samples

The Employee Lookup Application samples illustrate different types of applications you can create using VisualAge COBOL, including the use of databases. The samples are variations of a single application. All have the same external behavior consisting of the same GUI.

Each application is supplied as a set of projects with all the required source code. The projects are located in the Samples folder, which is located in the VisualAge COBOL folder.

Note: The executable files themselves are not supplied.

The following applications are supplied:

- Employee Lookup intended to run on a single OS/2 workstation, using inline data (see “Sample Project 1” on page 140).
- Employee Lookup intended to run on a single OS/2 workstation, using inline data, and using object-oriented (OO) COBOL methodology (see “Sample Project 2” on page 141).
- Employee Lookup intended to run on a single OS/2 workstation, using DB2 for OS/2 relational data, and using OO COBOL methodology (see “Sample Project 3” on page 141).
- Employee Lookup with a client intended to run on an OS/2 system and a server intended to run on an MVS CICS/ESA system. The application uses inline data (see “Sample Project 4” on page 143).
- Employee Lookup with a client intended to run on one OS/2 system and a server intended to run on another OS/2 system. The application uses inline data (see “Sample Project 5” on page 144).
- Employee Lookup with a client intended to run on an OS/2 workstation using VSAM data (see “Sample Project 6” on page 146).

Employee Lookup Application Description

The sample Employee Lookup application permits users to display employee information by way of a GUI running on a workstation. Which employees are displayed is based on search criteria shown below.

VisualAge COBOL Sample Applications

The following information is displayed for each employee:

- Last name
- First name
- Middle initial
- Department
- Phone number (in the format aaa-xxx-xxxx where aaa is the area code)
- Date of hire (in the format mm/dd/yy)
- Number of full years of service with the company

Note: All of the employee names and phone numbers that appear in the application are fictitious.

The search criteria are one of the following:

- Display all employees
- Display those employees that exactly match one of the following search information:
 - Last name
 - Department
 - Last name and department
- Display those employees that partially match (whose leading characters match) one of the following search information:
 - Last name
 - Department
 - Last name and department

The application provides online help information on using the application, as well as context sensitive help for the GUI parts such as entry fields, output fields, radio buttons, and push buttons.

An error message window is displayed for invalid request, match not found, data base error, and communications error. Online help information for the messages is provided.

Sample Project 1

This is the Employee Lookup application with a GUI, intended to run on a single workstation. The application uses inline data.

This application consists of a GUI project named **Sample Project 1**. The GUI project contains all the source parts associated with the GUI, including the COBOL source file.

Nested within the GUI project are two COBOL projects named **Search Logic Subroutine** and **Service Calculation Subroutine**. These projects contain the parts for the two subroutines that are called by the GUI program.

VisualAge COBOL Sample Applications

Building and Running the Project

To build *Sample Project 1*:

1. Double-click on the **VisualAge COBOL** icon.
2. Double-click on the **Samples** folder.
3. Double-click on the **Sample Project 1** project.
4. In the *Sample Project 1 - Icon View*, select the **Project** menu bar choice and then select **BUILD**.

The output from the BUILD is displayed in the project monitor area.

To run *Sample Project 1*:

1. In the *Sample Project 1 - Icon View*, select the **Project** menu bar choice and then select **RUN**.

Sample Project 2

This is the Employee Lookup application with a GUI, intended to run on a single workstation. The application uses inline data and object-oriented COBOL methodology.

This application consists of a GUI project named **Sample Project 2**. The GUI project contains all the source parts associated with the GUI, including the COBOL source file.

Nested within the GUI project is a COBOL project named **Data Base Class Program**. This project contains the parts for the object-oriented data base class program. This class program contains the methods that are invoked by the GUI program.

Building and Running the Project

To build *Sample Project 2*:

1. Double-click on the **VisualAge COBOL** icon.
2. Double-click on the **Samples** folder.
3. Double-click on the **Sample Project 2** project.
4. In the *Sample Project 2 - Icon View*, select the **Project** menu bar choice and then select **BUILD**.

The output from the BUILD is displayed in the project monitor area.

To run *Sample Project 2*:

1. In the *Sample Project 2 - Icon View*, select the **Project** menu bar choice and then select **RUN**.

Sample Project 3

This is the Employee Lookup application with a GUI, intended to run on a single workstation. The application uses DB2 for OS/2 relational data and object-oriented COBOL methodology.

VisualAge COBOL Sample Applications

This application consists of a GUI project named **Sample Project 3**. The GUI project contains all the source parts associated with the GUI, including the COBOL source file.

Nested within the GUI project is a COBOL project named **Data Base Class Program**. This project contains the parts for the object-oriented data base class program. This class program contains the methods that are invoked by the GUI program.

Building and Running the Project

To build and run *Sample Project 3*, you need to have the following installed:

- IBM DB2 for OS/2 Version 2.1 Single-User

To build *Sample Project 3*:

1. Create the database and table.

Before building *Sample Project 3*, you need to create the DB2 database, table, and data that this application requires. A procedure, IWZZ3CR.CMD, is provided to do this. You only need run this procedure one time. If you run the procedure again, you will need to rebuild the project again.

To create the DB2 database, table, and data, from an OS/2 window enter:

```
%ICOBDIR%\SAMPLES\IWZZ3\IWZZ3CR.CMD
```

2. Start the IBM DB2 for OS/2 product, if it has not already been started. To start DB2 for OS/2, from an OS/2 window enter:

```
DB2START
```

You can also start DB2 for OS/2 by double-clicking on the **IBM DATABASE 2** folder and then double-clicking on the **Start DB2** icon.

3. Double-click on the **VisualAge COBOL** icon.
4. Double-click on the **Samples** folder.
5. Double-click on the **Sample Project 3** project.
6. In the *Sample Project 3 - Icon View*, select the **Project** menu bar choice and then select **BUILD**.

The output from the BUILD is displayed in the project monitor area.

To run *Sample Project 3*:

1. Start the IBM DB2 for OS/2 product, if it has not already been started. To start DB2 for OS/2, from an OS/2 window enter:

```
DB2START
```

You can also start DB2 for OS/2 by double-clicking on the **IBM DATABASE 2** folder and then double-clicking on the **Start DB2** icon.

2. In the *Sample Project 3 - Icon View*, select the **Project** menu bar choice and then select **RUN**.

VisualAge COBOL Sample Applications

Sample Project 4

This is the Employee Lookup application with a GUI that has a client intended to run on an OS/2 system and a server intended to run on an MVS CICS/ESA system. The application uses inline data.

This application is contained in an OS/2 folder named **Sample Project 4**. This folder is used for organizational purposes.

This folder contains a GUI project named **Client GUI** and a COBOL project named **Server**. The GUI project contains all the source parts associated with the client GUI, including the COBOL source file.

Nested within the GUI project is a COBOL project named **Service Calculation Subroutine**. This project contains the parts for the subroutine that is called by the GUI program.

The COBOL project named **Server** contains the parts for the server that is called by the GUI program by way of the CICS OS/2 ECI facility.

The **Server** project inherits from the COBOL MVS Master Project. This inheritance will only show if you have installed the Remote Edit/Compile (beta) component. The parts in this project are intended to be placed on the MVS host.

Building and Running the Project

To build and run *Sample Project 4*, you need to have the following installed:

- The Transaction Assistant component of the VisualAge COBOL product.
- CICS Client for OS/2 Version 1 or CICS for OS/2 Version 2 distributed feature Client for OS/2

This product must be installed on the machine that is intended to be the client for the client/server application.

- A host (MVS) CICS system configured to be a server.

The CICS/ESA system programmer must define to the CICS/ESA DFHCVT (conversion table) an entry for the IWZZ4SV server program. This entry should indicate that the entire CICS COMMAREA needs to be translated from ASCII to EBCDIC. The COMMAREA is 584 bytes long.

Before building *Sample Project 4*, the following files need to be uploaded to the host (MVS) system that is to be the server machine.

- **d:\ibmcobol\SAMPLES\IWZZ4\IWZZ4SI.CPY**

where d:\ibmcobol represents the drive and directory where the VisualAge COBOL *Visual Tools* were installed.

The file should be placed in a PDS used for COBOL copybooks. The PDS member name should be IWZZ4SI.

- **d:\ibmcobol\SAMPLES\IWZZ4\IWZZ4SV\IWZZ4SV.CBL**

VisualAge COBOL Sample Applications

where d:\ibmcobol represents the drive and directory where the VisualAge COBOL *Visual Tools* were installed.

The file should be placed in a PDS used for COBOL source. The PDS member name should be IWZZ4SV.

To build *Sample Project 4*:

1. At the MVS host compile and link the IWZZ4SV server program. The PDS containing the copybook file IWZZ4SI needs to be one of the SYSLIB data sets. Since the IWZZ4SV program is run under CICS, it needs to be linked into the appropriate CICS link library.

To compile and link the IWZZ4SV server, you can either use your existing procedures for compile and link of CICS programs or use the VisualAge COBOL Remote Edit/Compile component. To use the Remote Edit/Compile component, see the **Task Helper** topic *Working with Host (MVS) Applications* for details.

2. Double-click on the **VisualAge COBOL** icon.
3. Double-click on the **Samples** folder.
4. Double-click on the **Sample Project 4** folder.
5. Double-click on the **Client GUI** project.
6. In the *Client GUI - Icon View*, select the **Project** menu bar choice and then select **BUILD**.

The output from the BUILD action is displayed in the project monitor area.

To run *Sample Project 4*:

1. Start either the IBM CICS Client for OS/2 or the CICS for OS/2 Version 2 distributed feature Client for OS/2.

To start the IBM CICS Client for OS/2, double-click on the **IBM CICS Client for OS/2** folder and then double-click on the **Start Client** icon.

2. In the *Client GUI - Icon View*, select the **Project** menu bar choice and then select **RUN**.

Sample Project 5

This is the Employee Lookup application with a GUI that has a client intended to run on one OS/2 system and a server intended to run on another OS/2 system. The application uses inline data.

This application is contained in an OS/2 folder **Sample Project 5**. This folder is used for organizational purposes.

This folder contains a GUI project named **Client GUI** and a COBOL project named **Server**. The GUI project contains all the source parts associated with the client GUI, including the COBOL source file.

VisualAge COBOL Sample Applications

Nested within the GUI project is a COBOL project named **Service Calculation Subroutine**. This project contains the parts for the subroutine that is called by the GUI program.

The COBOL project named **Server** contains the parts for the server that is called by the GUI program by way of the CICS OS/2 ECI facility.

Building and Running the Project

To build and run *Sample Project 5*, you need to have the following installed:

- The Transaction Assistant component of the VisualAge COBOL product.
- CICS Client for OS/2 Version 1 or CICS for OS/2 Version 2 distributed feature Client for OS/2

This product must be installed on the machine that is intended to be the client for the client/server application.

- CICS for OS/2 Version 3.0

The CICS for OS/2 Version 3.0 product is available in limited Beta until its United States general availability. The Beta code is shipped on the IBM VisualAge for COBOL for OS/2 CD-ROM.

This product must be installed on the machine that is intended to be the server for the client/server application. You may choose to have the server machine be the same as the client machine. The CICS System Initialization Table (SIT) needs to be updated to define the CICS system as a server. In addition, if the server is on a different machine as the client, LAN setup may be required. Refer to the online CICS *Intercommunication* documentation for details.

To build *Sample Project 5*:

1. Double-click on the **VisualAge COBOL** icon.
2. Double-click on the **Samples** folder.
3. Double-click on the **Sample Project 5** folder.
4. Double-click on the **Server** project.
5. In the *Server - Icon View*, select the **Project** menu bar choice and then select **Build**.
The output from the Build action is displayed in the project monitor area.
6. Close the **Server** project.
7. Do one of the following based on whether the server machine is the same as the client machine:
 - If the server machine is the same as the client machine, copy the IWZZ5SV.DLL file to the CICS run-time directory. From an OS/2 command window, enter all on one line :

```
COPY %ICODIR%\SAMPLES\IWZZ5\IWZZ5SV\IWZZ5SV.DLL  
d:\cics300\RUNTIME
```

VisualAge COBOL Sample Applications

where d:\cics300 represents the drive and directory where CICS for OS/2 Version 3.0 was installed.

- If the server machine is not the same as the client machine then you need to package the server and the VisualAge COBOL run-time and install them on the server machine. Refer to the section *Distributing the OS/2 Application* in the **Task Helper** topic *Building an OS/2 Application* for details on packaging and installing the server.

8. Double-click on the **Client GUI** project.
9. In the *Client GUI - Icon View*, select the **Project** menu bar choice and then select **BUILD**.

The output from the BUILD action is displayed in the project monitor area.

To run *Sample Project 5*:

1. Start the CICS for OS/2 Version 3.0 product on the server machine.
2. Start either the IBM CICS Client for OS/2 or the CICS for OS/2 Version 2 distributed feature Client for OS/2.

To start the IBM CICS Client for OS/2 double-click on the **IBM CICS Client for OS/2** folder and then double-click on the **Start Client** icon.

3. In the *Client GUI - Icon View*, select the **Project** menu bar choice and then select **RUN**.

Sample Project 6

This is the Employee Lookup application intended to run on a single OS/2 workstation using VSAM data. When invoked as supplied, the application accesses local VSAM data on the same workstation. Through environmental setup, including APPC communications setup, the application can access MVS VSAM data. This does not require changing the application or rebuilding the application.

This application consists of a GUI project named **Sample Project 6**. The GUI project contains all the source parts associated with the client GUI, including the COBOL source file.

Nested within the GUI project are two COBOL projects named **Search Logic Subroutine** and **Service Calculation Subroutine**. These projects contain the parts for the two subroutines called by the GUI program.

Building and Running the Project

To build *Sample Project 6*:

1. Double-click on the **VisualAge COBOL** icon.
2. Double-click on the **Samples** folder.
3. Double-click on the **Sample Project 6** project.
4. In the *Sample Project 6 - Icon View*, select the **Project** menu bar choice and then select **BUILD**.

VisualAge COBOL Sample Applications

The output from the BUILD is displayed in the project monitor area.

To run *Sample Project 6*:

1. In the *Sample Project 6 - Icon View*, select the **Project** menu bar choice and then select **RUN**.

SMARTdata Utilities Samples

The SMARTdata Utilities provide

- A local record level access method provided by VSAM for the workstation. This allows you to have sequential, direct, and keyed files on your local system.
- Remote access to files residing on MVS, OS/400, and CICS systems using the same VSAM interface used to access local files. On MVS, this includes access to sequential access method (SAM) files and partitioned data set extended (PDSE) members. Remote access is integrated with a data conversion engine that allows you to view even complicated record structures from remote systems in data formats supported by your local machine.
- A general purpose data conversion engine for more complex conversion tasks.
- An industrial strength sort, merge, copy, and extract package, called SMARTsort.

The SMARTdata Utilities samples illustrate the following tasks:

- Using VSAM for the workstation for local and remote data access, including customizing data description and conversion for transparent remote data access
- Calling the data conversion utility directly from within an application.
- Using SMARTsort to sort and filter data in a file

VSAM for the Workstation Samples

These samples illustrate configuration and use of VSAM for the workstation for remote and local data access. The following samples are located in the **SDU** folder within the **SAMPLES** folder:

- DUBSAMP.C (and associated files) is a sample program written in C that demonstrates some basic record I/O using the VSAM APIs directly. The sample program is provided for users who wish to employ the VSAM interface from a language that is not already integrated with VSAM. The sample program will access both local and remote files, since the same interface is used in both cases. The header files used by DUBSAMP.C are located in the **INCLUDE** folder.

A COBOL sample illustrating local data access to VSAM can be found in **Sample Project 6**, as part of the Employee Lookup Application Samples.

- OS2.SNA, OS400.SNA, OS4680.SNA, and CICS.SNA are samples describing network definitions used when configuring the remote file access portion of VSAM (called Distributed FileManager.)
- STARTDFM.CMD is a sample command file to start the Distributed FileManager. STARTUP.CMD provides sample statements that can be included in an OS/2

VisualAge COBOL Sample Applications

STARTUP.CMD file, to start the Distributed FileManager automatically at OS/2 start up time.

- SAMPASCI.ADL, SAMPBASE.ADL, SAMPEBCD.ADL, and SAMPVIEW.ADL are sample data conversion descriptions. A data conversion description is used by the Distributed FileManager to determine what conversions (if any) must be done to data in a remote file in order to present it properly to the requesting program.
- EHNXNMP.C (and associated files) is a sample name mapping exit routine written in C.

Data Conversion Utility Samples

Data Description and Conversion APIs provide the ability to customize, extend, and manage data conversion capabilities from within an application. The following samples are located in the **SDU** folder within the **SAMPLES** folder:

- IWZZSS1.CBL is a sample COBOL program that demonstrates the use of the Data Description and Conversion Parse and Generate functions and Conversion Plan Builder. IWZZSS1.ADL is input to this program.
SAMPLE1.C (and associated files) provides the same sample written in C.
- IWZZSS2.CBL is a sample COBOL program that demonstrates the use of the Conversion Plan Executor component of Data Description and Conversion. As input, it uses the conversion plans created by program IWZZSS1.
SAMPLE2.C (and associated files) provides the same sample written in C.

SMARTsort Samples

Two COBOL samples are provided in the **SMRTSORT** folder, located within the **SAMPLES** folder:

- SAMPLE7.CBL demonstrates how SMARTsort can be used to filter data from a file.
- SAMPLE8.CBL demonstrates how SMARTsort can be used to create a sorted file consisting of records that have been restructured from the original file.

Several samples written in C are also available in this folder.

The data files used by these samples can be found in the **DATA** folder located within the **SMRTSORT** folder.

Appendix C. Configuring APPC Communications

Using the SMARTdata UTILITIES (SdU) component or the Remote Edit/Compile component (which includes the Remote PWS Debug Tool) to access host data requires communications to be configured at the workstation and at the host. You can use IBM Debug Tool to debug applications residing on the host directly from your programmable workstation (PWS). This chapter describes configuring communications for cooperative sessions between the workstation and the host using advanced program-to-program communication (APPC) protocol. The term Remote PWS Debug Tool is used when discussing debugging host applications from a PWS.

Note: The Remote Edit/Compile component is provided only at beta level.

You must use APPC if you plan to use either the SdU component or the Remote Edit/Compile component. In addition to support using APPC (LU6.2) communications, the Remote PWS Debug Tool is supported using LU2.0 communications as well. Using LU2.0 with the Remote PWS Debug Tool requires no setup beyond having a CM/2 3270 emulator session. However, there is significant function available with APPC that is not provided with LU2 and, if you are interested in cooperative debugging, APPC is recommended.

Configuring APPC cooperative sessions requires coordination between workstation and host administrators. Configuring Communications Manager/2 (CM/2) at the workstation requires:

- Defining the workstation to the network.
- Defining a link from the workstation to the host, or an intermediate APPN node (our examples will show the link being defined to the host). The terms *link* and *connection* are used interchangeably in this chapter.
- Defining the workstation as a client for APPC sessions
- Defining the workstation as a server for Remote PWS Debug Tool APPC sessions

The figures in this chapter illustrate workstation configuration windows from Communications Manager/2 Version 1.11. If you need additional information for any Communications Manager window discussed in this section, press PF1 to obtain help, refer to the following publications, or contact your IBM representative:

- *CM/2 Information and Planning Guide*, SC31-7007
- *CM/2 Installation and Configuration Guide*, SC31-7169

Configuring the host for APPC requires:

- Defining APPC/MVS Server facilities for cooperative sessions
- Defining VTAM and NCP definitions, (if communications are through an NCP)
- Optionally defining client facilities within CICS for Remote PWS Debug Tool sessions

The assumed network in this chapter is a token-ring LAN connected to a host system via a 3745 communications controller equipped with a token-ring adapter. NCP and

APPC at CM/2

VTAM definitions are only examples and will vary with network characteristics. If you need additional assistance to configure your network, see the appropriate following publications, or contact your IBM representative:

- *MVS/ESA Planning: APPC Management*, GC28-1110
- *VTAM Resource Definition Reference*, SC31-6438
- *VTAM Network Implementation Guide*, SC31-6434
- *NCP Resource Definition Reference*, SC30-3448
- *CICS/ESA: Resource Definition Guide*, SC33-1166
- *CICS/ESA: System Definition Guide*, SC33-0664

Approaching the Task of Configuring Communications

While APPC does not require 3270 emulator sessions at the workstation, it is assumed that you want to run 3270 emulators in parallel with APPC sessions. See *Communication Manager Configuration Guide* if you do not already have CM/2 emulator sessions defined at the workstation.

APPC configuration requires APPC/MVS, CM/2 - OS/2, and possibly CICS to be configured. This chapter describes configuration requirements for all three platforms.

Configuring APPC across computing platforms requires sharing information between platform administrators. After VisualAge COBOL has been installed, APPC configuration should begin with each administrator completing definitions required for his or her respective platform. For a specific platform, some configuration parameters will originate from that platform, while other parameters will derive from a partner platform.

To simplify the configuration process and facilitate information sharing, each platform has a table of configuration variables that should be defined prior to configuration. These definitions can be used to generate worksheets allowing you to perform configuration on the respective platforms.

Configuring for APPC Communications at CM/2

This section describes how to configure APPC for a CM/2 workstation that is connected to a token-ring Local Area Network (LAN) and the required workstation software.

Prerequisites

- OS/2 Version 2.11 or later
- Communications Manager/2 Version 1.11 or later

Terminology

Below is a table of standard APPC terms and the equivalent CM/2 terms.

CM/2 Configuration Variables

<i>Standard Term</i>	<i>CM/2 Term</i>
Network Name	Network Name
LU Name	Local Node Name
Partner LU Name	Partner Node Name
Local LAN Address	Local MAC Address
Adjacent LAN Address	LAN Destination Address

Communications Manager/2 Configuration Variables

Use the following table to create a worksheet that defines the listed symbols. These symbols and their corresponding values are used to configure CM/2 for communication with an APPC/MVS and/or CICS host. The worksheet consists of two parts:

1. Symbols that must be matched with a partner platform (for example, MVS/APPC or CICS) are under the heading **NETWORK VALUES**.
2. Symbols that are local to the workstation are under the heading **LOCAL VALUES**.

Note: The values defined in Table 3 are compatible with an appropriately configured APPC/MVS or CICS host. A workstation configured with these values will support Edit/Compile sessions with APPC/MVS and/or Remote PWS Debug Tool sessions with APPC/MVS and/or CICS. In this configuration, it is assumed that CICS and APPC/MVS both reside on the same MVS host. If this is not the case and you require assistance, contact your IBM representative.

Table 3 (Page 1 of 4). Values Required to Configure CM/2 for APPC

Symbols	How to determine the value for this symbol	Example value	Fill in your value here
NETWORK VALUES			
LUNAME	This symbol refers to the LU name of the workstation that you are defining to the network. The value of this symbol should be supplied by the appropriate network administrator. Based on the level of VTAM and the use of APPN, the VTAM or LAN administrator will know the value of this symbol.	ELNQF0EA	
CPNAME	This symbol defines the control point name of the workstation. For workstations that define a single APPC LU, this symbol will default to the LUNAME value. The VTAM administrator can use this value to identify this workstation during session initialization (see the PU definition in the section titled "Defining the 3745 Attached LAN to VTAM" on page 180 for more information).	ELNQF0EA	

CM/2 Configuration Variables

<i>Table 3 (Page 2 of 4). Values Required to Configure CM/2 for APPC</i>			
Symbols	How to determine the value for this symbol	Example value	Fill in your value here
NODEID	This is the value as defined in VTAM for IDBLK (05D) and IDNUM. This value will be provided by the host VTAM administrator if VTAM uses IDBLK and IDNUM to identify this workstation during session initialization (as opposed to CPNAME). See the values of IDBLK and IDNUM in Table 4 on page 171.	05D00F0E	
NETWORK	This symbol identifies the network that the workstation is being defined within. If the workstation belongs to the same network as the host system, this symbol should be supplied by your VTAM administrator. See the value of NETNAME in Table 4 on page 171.	CAIBMOML	
PARTNERCP	This symbol defines the partner control point (CP) name. Assuming the partner CP is the host, this value should be provided by the VTAM administrator. See the value of LOCALCP in Table 4 on page 171. PARTNERCP is used to define the host connection to CM/2.	OMA	
PARTNERLU	This symbol identifies the APPC/MVS LU to the workstation. This value should be supplied by the APPC/MVS or VTAM administrator of the host system to which you are connecting. The host administrator should supply the name of the APPC/MVS LU that is designated for use with VisualAge COBOL. See the value of LUNAME in Table 4 on page 171. This symbol is used in defining Common Programming Interface-Communications (CPI-C) side information in defining a host connection.	SA07APPC	
PARTNERNET	This symbol is the name of the network in which your MVS host resides. This symbol should be supplied by the VTAM administrator of the host to which you are connecting. See the value of NETNAME in Table 4 on page 171.	CAIBMOML	
LOCALPU	If <i>host focal support</i> is defined for an associated connection, this value defaults to CPNAME . If <i>host focal support</i> is not defined for a connection, this value can be any name you choose as long as it uniquely defines the PU for a connection within CM/2.	ELNQF0EA	

CM/2 Configuration Variables

<i>Table 3 (Page 3 of 4). Values Required to Configure CM/2 for APPC</i>			
Symbols	How to determine the value for this symbol	Example value	Fill in your value here
LINKNAME	This symbol is used to reference a connection or link that is defined to either the host system or an intermediary node. When defining a new link, this symbol can be assigned a value of your choice (for example, HOST0002).	HOST0002	
LANADDRESS	This symbol is for the LAN Address of the partner computer (or the LAN Address of a network node, if the link is being made to a network node). In the sample network described in this chapter, LANADDRESS is the token-ring address in the 3745 to which the LAN is attached. This address should be supplied by the LAN or NCP administrator. See the value of NCPLANADDR in Table 4 on page 171.	400011528909	
REMOTETP	This symbol is for the transaction program residing at APPC/MVS. It is transmitted across the network to APPC/MVS to initiate a session with the server. This symbol is used to define workstation CPI-C side information entries for sessions. Use the value COBOLVS_MVSLU62_EC_SERVER as the value for REMOTETP .	See symbol value in adjacent column	
LOCALTP	This is the symbol for the transaction program that resides on the workstation. It is transmitted across the network from APPC/MVS or CICS in order to initiate a session with the Remote PWS Debug Tool server. For the Remote PWS Debug Tool, specify COBVSDDT as the value for LOCALTP .	COBVSDDT	
TPPATH	This is the path and filename for the executable program that is defined by LOCALTP . For the Remote PWS Debug Tool, use the following value if you did not install VisualAge COBOL into a different high-level directory: drive:\IBMCOBOL\BIN\EQACEL62.EXE	See path in adjacent column	
MODE	This symbol is for the mode name. #INTER is recommended. This value should correspond to MODE at the host. For more information about MODE at the host, see Table 4 on page 171. This symbol is used to define CPI-C side information entries at the workstation for sessions.	#INTER	
LOCAL VALUES			

Performing CM/2 APPC Configuration

Symbols	How to determine the value for this symbol	Example value	Fill in your value here
NODETYPE	This symbol is for the APPN node type. This should be EN configured as an end node, or NN configured as a network node. Use the value EN unless you are using APPN and this workstation is a network node.	EN	
LUALIAS	This symbol is for the Local LU Alias. It designates a nickname for the Local LU. Alias names are case-sensitive. Simply using the value of LUNAME is adequate.	ELNQF0EA	
PLUALIAS	This symbol is for the Partner LU Alias. It designates a nickname for the Partner LU. Alias names are case-sensitive.	TLBA07ME	
SYMDEST	This symbol is for the CPI-C side information symbolic destination name. It is used locally on the OS/2 machine to refer to a CPI-C side information entry that identifies the host server. The entry contains: <ul style="list-style-type: none"> • TP name (defined by REMOTETP) • LU name (defined by PARTNERLU) • MODE (defined by MODE) Note: This value can be specified in the Servers window.	COBOLVS	

Performing CM/2 APPC Configuration

This example assumes that CM/2 configuration files already exist. In other words, it is assumed that you have already installed CM/2 and only require APPC configuration. If this is not the case, install CM/2 as described in *CM/2 Installation and Configuration Guide* and resume at this point. Prior to APPC configuration, it is recommended that you copy your default CFG and NDF files (usually located in the C:\CMLIB directory) to another set of files that you can name to CVSCFG. For example, assuming that your default configuration is named BASIC, move to the C:\CMLIB directory and then type on an OS/2 command line:

```
COPY BASIC.* CVSCFG.*
```

This will create a copy of the various files that comprise your default configuration. This file will be modified in the following steps to contain APPC definitions while leaving other existing definitions intact. If you are modifying an existing configuration, as is the case in the example that follows, some definitions from your worksheet might already exist and need not be supplied by you.

Perform the following steps to modify the configuration CVSCFG:

Selecting a Configuration

Selecting the CVSCFG Configuration File

1. Double-click on the CM/2 icon.
2. Double-click on the CM/2 Setup icon. The **Communications Manager Setup** window appears, as shown in Figure 68.

Note: The CM/2 windows shown in the examples are taken from CM/2 Version 1.11. If you are using a newer version of CM/2, the windows might have a slightly different layout.

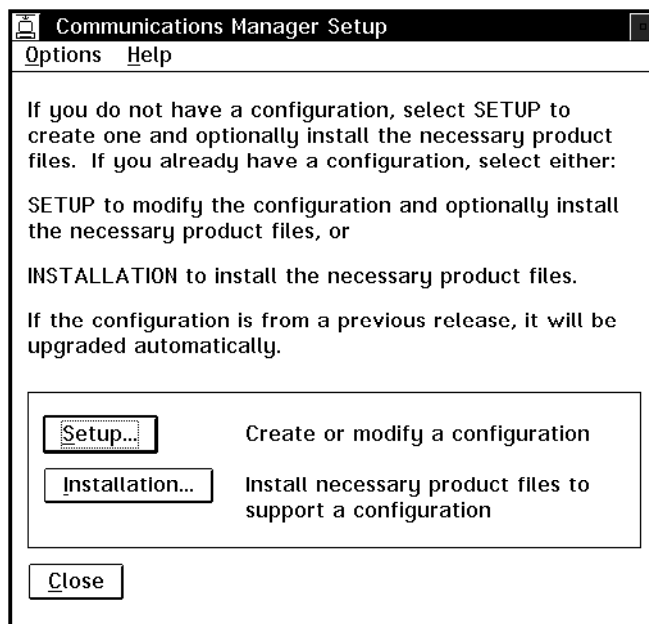


Figure 68. Communications Manager Setup Window

3. Select the **Setup** push button. The **Open Configuration** window appears, as shown in Figure 69 on page 156.

Selecting a Configuration

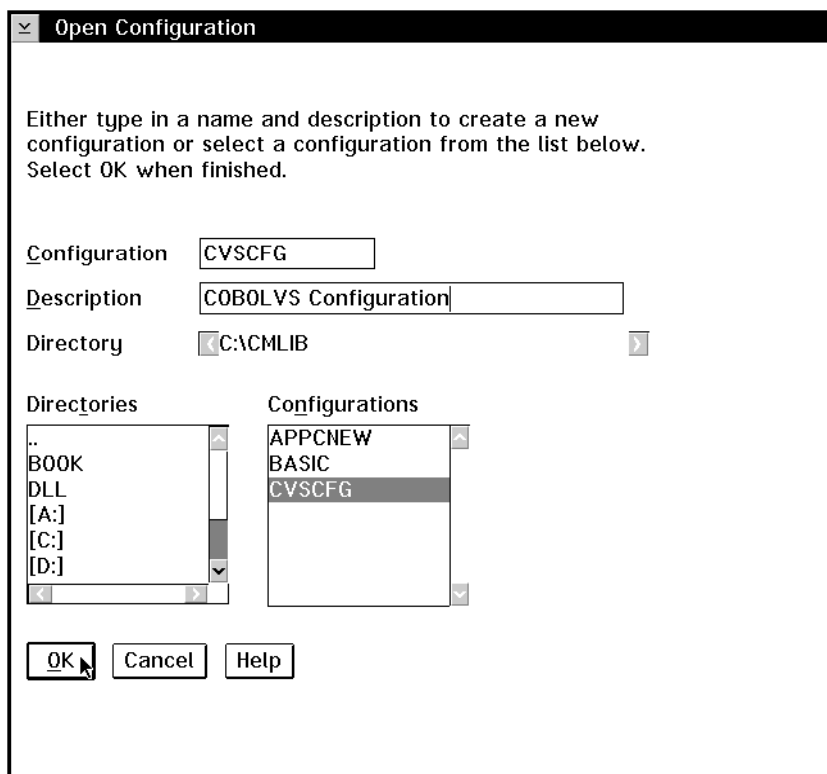


Figure 69. Open Configuration Window

4. Select CVSCFG from the **Configurations** list.
5. Select the **OK** push button.
6. When prompted, select the **Yes** push button to confirm that CVSCFG will be used for this workstation.
7. The **Communications Manager Configuration Definition - CVSCFG** window appears, as shown in Figure 70 on page 157. Make sure that you have selected the **Additional definitions** radio button; otherwise, the window might appear different from the window in the figure.

Defining to the Network

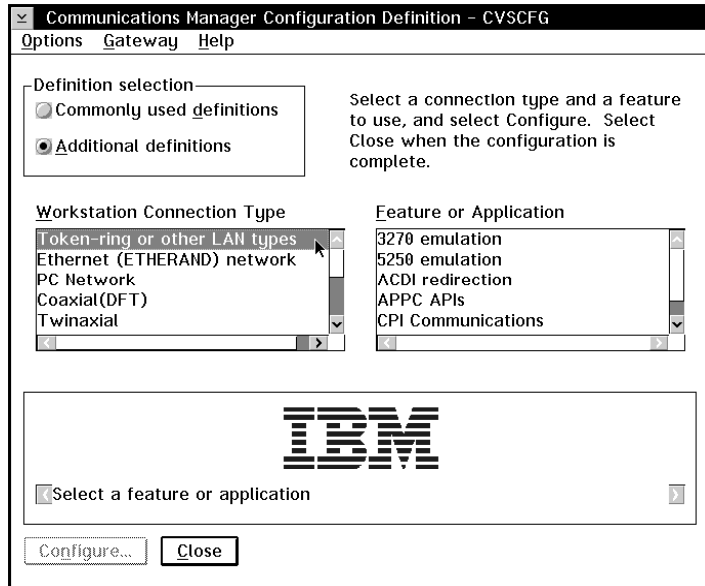


Figure 70. Communications Manager Configuration Definition - CVSCFG Window

Defining your Workstation to the Network

Before the workstation can communicate within a network, it must be recognized by that network. This means that the workstation must be properly defined in the **Communications Manager Local Node Characteristics** window. If the workstation has already been defined to the network, verify that the definitions are complete and consistent with the worksheet values. Otherwise, perform the following steps to define these values:

1. Select the option **Token-ring or other LAN types**.
2. Select **APPC APIs** from the **Feature or Application** list.
3. After selecting **APPC APIs**, the **Communications Manager Configuration Definition - CVSCFG** window, shown in Figure 71 on page 158, changes as a result of your choice.

Defining to the Network

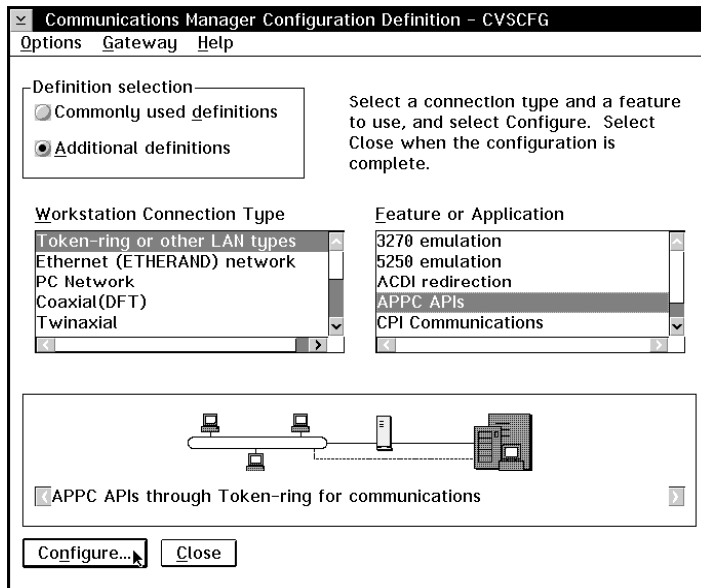


Figure 71. Communications Manager Configuration Definition - CVSCFG Window

4. Select the **Configure** push button.
5. The **APPC APIs through Token-ring** window appears, as shown in Figure 72.

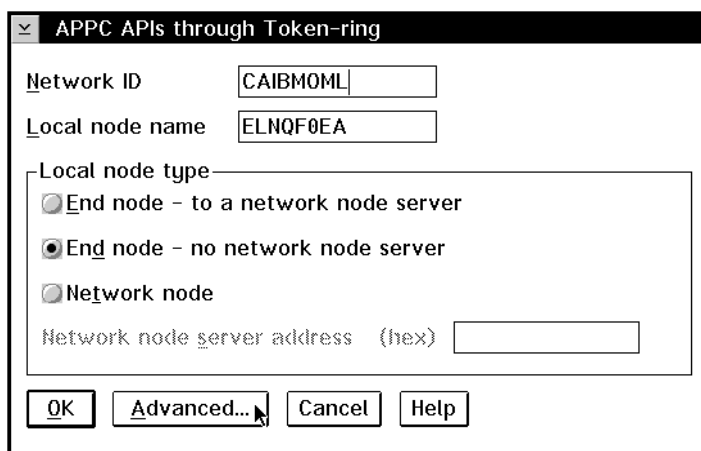


Figure 72. APPC APIs through Token-ring

6. In the **Network ID** field, type the **NETWORK** value.
7. In the **Local node name** field, type the **LUNAME** value.
8. For the **Local node type**, select **End node – no network node server**. This allows you to configure directly to the token-ring at the communications controller.

Defining to the Network

If you want to connect your host through an APPN network node, select **End node – to a network node server**. Our example does not address this form of connection.

9. Select the **Advanced** push button.
10. The **Communications Manager Profile List** window appears, as shown in Figure 73.

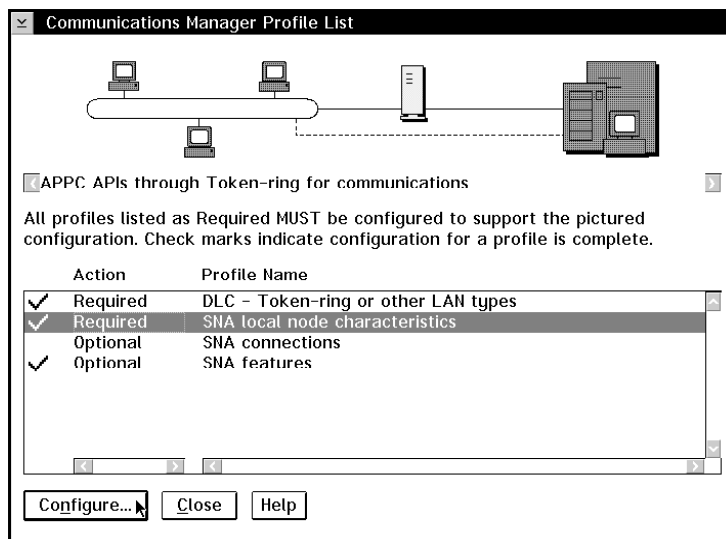


Figure 73. Communications Manager Profile List Window

11. Select **SNA local node characteristics** from the **Profile Name** list.
12. Select the **Configure** push button.
13. The **Local Node Characteristics** window, shown in Figure 74, appears.

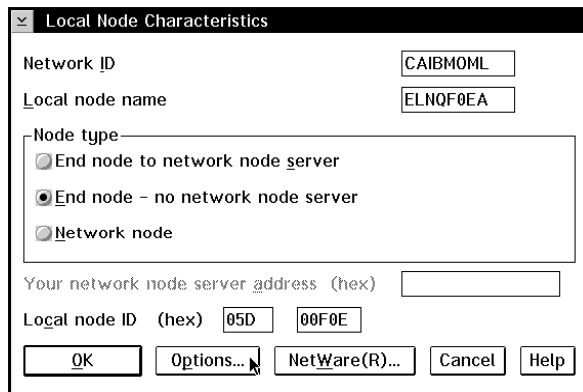


Figure 74. Local Node Characteristics Window

Defining a Connection

14. Confirm that the values you entered in the **APPC APIs through Token-ring** window are what you need for **Network ID**, **Local node name**, and **Local node type**.
15. Type the value of **NODEID** in the **Local node ID** field to identify the workstation with **NODEID** as opposed to **CPNAME**.
16. Select the **Options** push button.
17. The **Local Node Options** window appears, as shown in Figure 75.

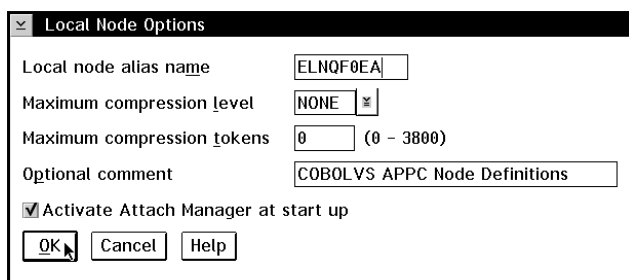


Figure 75. Local Node Options Window

18. In the **Local node alias name** field, type the **LUALIAS** value. This can be the same as the local node name (**LUNAME**).
19. Select **Activate Attach Manager at start up**.
20. Select the **OK** push button.
21. The **Local Node Characteristics** window appears, as shown in Figure 74 on page 159.
22. Select the **OK** push button.
23. The **Communications Manager Profile List** window appears, as shown in Figure 73 on page 159.

Defining a Connection

Before the workstation can communicate with the host, a corresponding connection must be defined. If the connection is already defined, verify that the definitions are complete and consistent with the worksheet values by stepping through the following windows. If you are defining a new connection, enter the field values as you move from one window to the next.

1. Select **SNA connections** from the **Profile Name** list.
2. Select the **Configure** push button.
3. The **Connections List** window appears, as shown in Figure 76 on page 161.

Defining a Connection

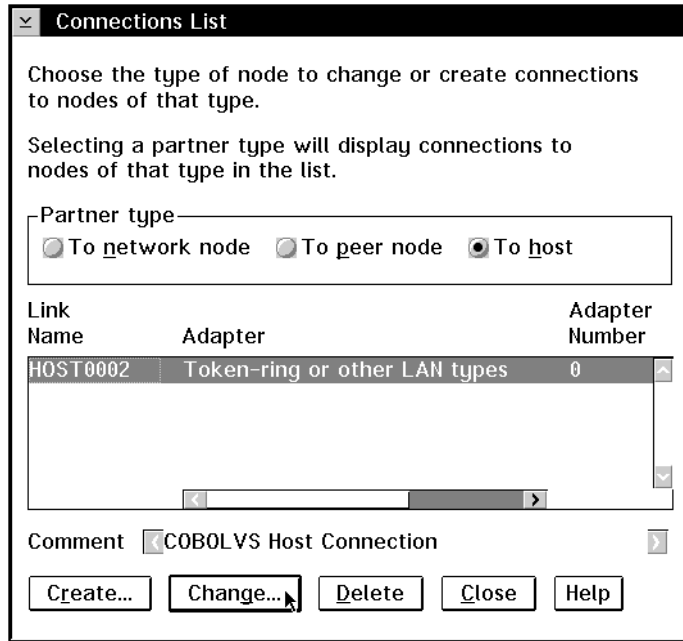


Figure 76. Connections List Window

4. Select **To host** as the partner to which you are connecting. Alternatively, you can connect to a network node, but in this example we are connecting to the host.
5. If a connection to your host already exists, select the connection corresponding to the value of **LINKNAME**. In our example, **LINKNAME** is **HOST0002**.
6. Assuming the connection to your host is already defined for emulators, select the **Change** push button. If no such connection exists, select the **Create** push button.
7. The **Adapter List** window, shown in Figure 77 on page 162, appears.

Defining a Connection

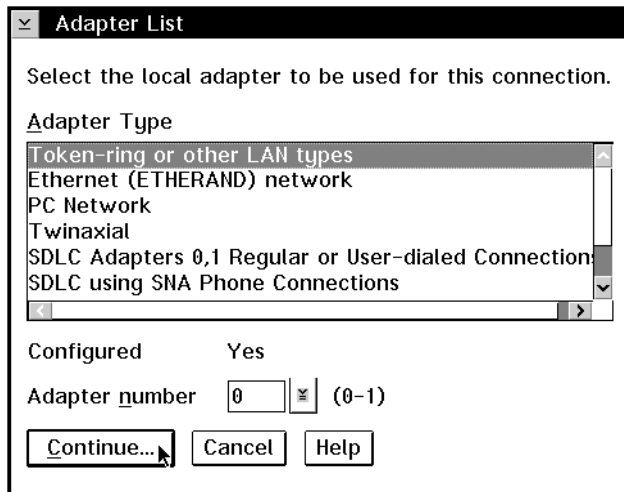


Figure 77. Adapter List Window

8. Since this example assumes a token-ring LAN, select **Token-ring or other LAN types**.
9. Select the **Continue** push button.
10. Since you selected **To host** in Figure 76 on page 161, the **Connection to a Host** window appears, as shown in Figure 78.

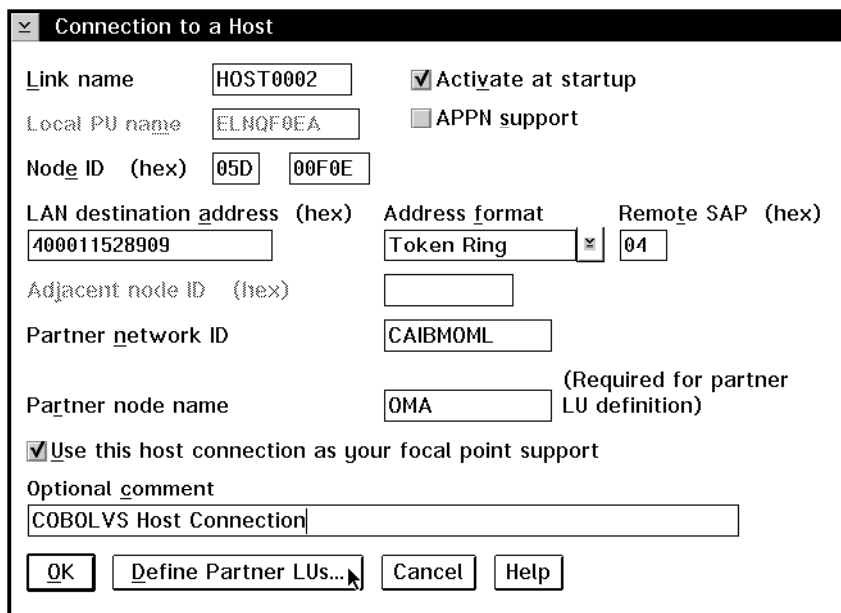


Figure 78. Connection to a Host Window

Defining the Partner LU

1. In the **Link Name** field, type the **LINKNAME** value.
2. Check the **Activate at startup** box to initialize this link when CM/2 is started.
3. For the **Local PU name** field, If this is the only host connection defined and **Use this host connection as your focal point support** (see step 8) is selected, this field defaults to **CPNAME**. Otherwise, enter the value of **LOCALPU**.
4. Assuming that VTAM recognizes this station by **NODEID** as opposed to **CPNAME**, specify its hexadecimal value here.
5. In the **LAN destination address** field, type the **LANADDRESS** value. For this type of connection, **LANADDRESS** refers to the token ring at the communication controller.
6. In the **Partner network ID** field, type the **PARTNERNET** value.
7. In the **Partner node name** field, type the **PARTNERCP** value.
8. If this is the only host connection defined from this workstation, check the **Use this host connection as your focal point support** box. If other host connections exist, only one can be so designated. To explore the alternative of not using host focal point services for a connection, see the bibliography for this section.

Defining the Partner LU

You can define the partner LU and an alias for the MVS/APPC LU. For example, you can specify that "HOSTLU" is an alias for **PARTNERNET.PARTNERLU**.

Skip this step if you selected **End node to network server** in the **Local Node Characteristics** window, shown in Figure 74 on page 159.

1. From the **Connection to a Host** window, shown in Figure 78 on page 162, select the **Define Partner LUs** push button.
2. The **Partner LUs** window appears, as shown in Figure 79 on page 164.

Defining the Partner LU

Partner LUs

To add a Partner LU, enter the LU name, alias, and comment. Then select Add.

To change a Partner LU, select an LU from the list, change the LU name, alias, and/or comment fields and select Change.

To delete a Partner LU, select an LU from the list and select Delete.

Network ID: CAIBM0ML

LU name: SA07APPC

Alias: TLBA07ME

Dependent partner LU

Partner LU is dependent

Uninterpreted name:

LU name	Alias
CAIBM0ML.SA07APPC	TLBA07ME

Delete

Optional comment: Define COBOLVS APPC/MVS LU and JES nodeid as alias

Add Change

OK Cancel Help

Figure 79. Partner LUs Window

3. In the **Network ID** field, type the value for **PARTNERNET**.
4. In the **LU name** field, type the value for **PARTNERLU**.
5. In the **Alias** field, type the value for **PLUALIAS**.

Note: An alias is case sensitive. "HOSTLU" is a different alias than "hostlu."

6. Select the **Add** push button.
7. Select the **OK** push button.

Returning to the Communication Manager Profile List

1. From the **Connection to a Host** window, shown in Figure 78 on page 162, select the **OK** push button.
2. The **Connections List** window appears, as shown in Figure 76 on page 161.
3. Select **Close**.
4. The **Communications Manager Profile List** window appears, as shown in Figure 73 on page 159.

Proceeding to the SNA Features List

1. From the **Communications Manager Profile List**, select **SNA features**.
2. Select the **Configure** push button.

Configuring to Run as a Client

3. The **SNA Features List** window appears.

Configuring to Run as a Client

To create a symbolic destination name as the target of a conversation, define a CPI-C side information entry as follows:

1. Select **CPI Communications Side Information** from the **Features** list in the **SNA Features List** window, as shown in Figure 80.

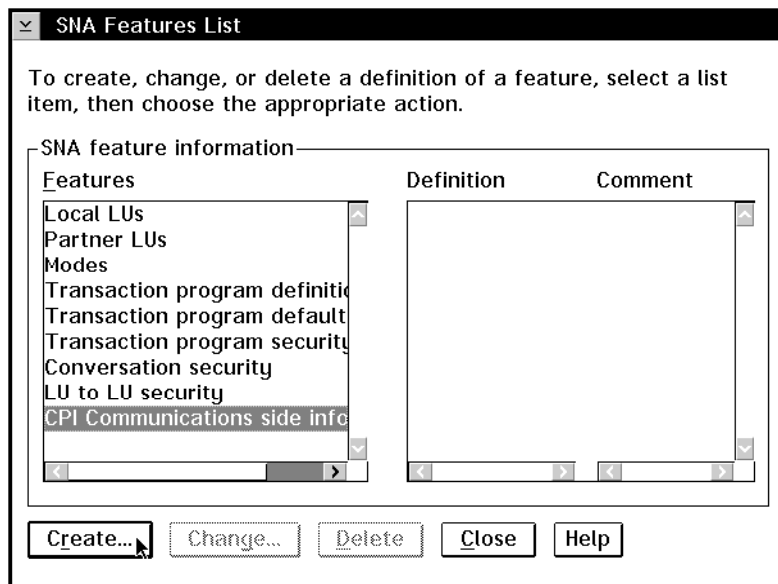


Figure 80. SNA Features List Window

2. Select the **Create** push button.
3. The **CPI Communications Side Information** window appears, as shown in Figure 81 on page 166.

Configuring to Run as a Client

The screenshot shows a dialog box titled "CPI Communications Side Information". It contains several input fields and radio buttons. The "Symbolic destination name" field is filled with "COBOLVS". Under "Partner LU", the "Alias" radio button is selected, and the field next to it contains "TBLA07ME". Under "Partner TP", the "Service TP" checkbox is unchecked, and the "TP name" field contains "COBOLVS_MVSLU62_EC_SERVER". Under "Security type", the "Program" radio button is selected. The "Mode name" field contains "#INTER". An "Optional comment" field contains the text "Create SymDestName to identify Host Server". At the bottom, there are three buttons: "Continue...", "Cancel", and "Help".

Figure 81. CPI Communications Side Information Window

4. In the **Symbolic destination name** field, type the **SYMDEST** value.
5. In the **Alias** field, type the **PLUALIAS** value.
6. In the **TP name** field, type the **REMOTETP** value.
7. Select **Program** as the **Security type**. This allows you to enter your TSO userid and the password on the MVS host where the server exists.
8. Select **#INTER** in the **Mode name** field.
9. Select the **Continue** push button. The **CPI Communications Program Security** window appears, as shown in Figure 82.

The screenshot shows a dialog box titled "CPI Communications Program Security". It contains a text prompt: "Enter your User ID, and then type your password twice for confirmation. Select OK." Below the prompt are three input fields: "User ID" with "TSCHARL", "Password" with "*****", and "Retype the password" with "*****". At the bottom, there are three buttons: "OK", "Cancel", and "Help".

Figure 82. CPI Communications Program Security Window

Configuring Remote PWS Debug Tool to Run as a Server

10. Type your host TSO id and password in the indicated fields. This is the id where your cooperative edit and compile sessions will be conducted.

Note:

As part of the communications setup, an encrypted version of your TSO password is stored on your workstation and is used to establish communications with MVS. Changing your TSO password also necessitates updating encrypted version stored at your workstation. If you attempt to establish a host session without updating this encrypted password, the following messages will appear in the **CODE PAM0031I - INFORMATION** message box:

No server object shown.

You may not have started a CODE conversation to the host.

This message can be the result of other communications errors, not just a password failure.

If you click on the message box **OK** push button, the attempt to establish host communications will be retried. If the failure was due to an incorrect password, multiple retries can result in your TSO access to your TSO userID being revoked, requiring an MVS security administrator's assistance for reinstatement.

Only after a certain number of retry attempts is the message box displayed. Therefore, it is possible that, by the time the message box displays, access to your TSO userID has already been revoked. The only way to prevent additional retries is to reboot your OS/2 system as soon as the message box is displayed. If using the **Ctrl+Alt+Delete** keys does not cause a reboot, power off and power on to reinitialize your workstation.

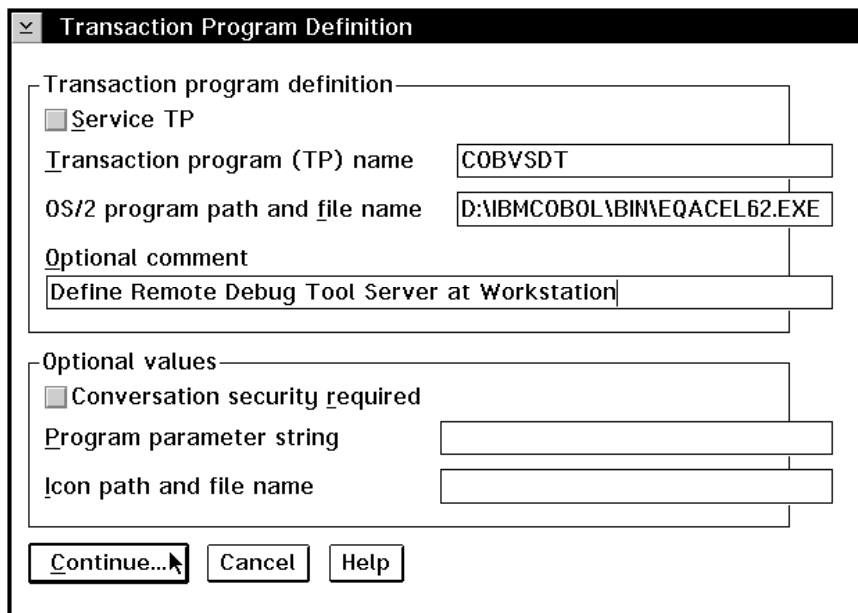
11. Select the **OK** push button.
12. The **SNA Features List** window appears, as shown in Figure 80 on page 165.

Configuring the Remote PWS Debug Tool to Run as a Server

To configure a server for the Remote PWS Debug Tool:

1. Select **Transaction Program Definition** from the **Features** list.
2. Select the **Create** push button. The **Transaction Program Definitions** window appears, as shown in Figure 83 on page 168.

Configuring Remote PWS Debug Tool to Run as a Server

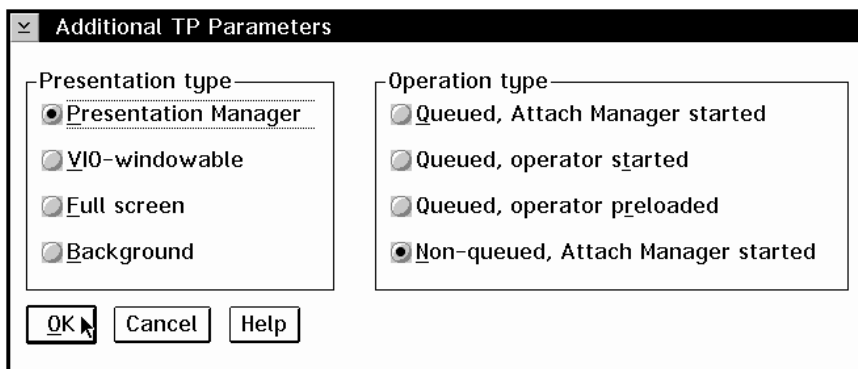


The dialog box is titled "Transaction Program Definition". It contains the following fields and options:

- Service TP
- Transaction program (TP) name: COBVSDT
- OS/2 program path and file name: D:\IBMCOBOL\BIN\EQACEL62.EXE
- Optional comment: Define Remote Debug Tool Server at Workstation
- Optional values section:
 - Conversation security required
 - Program parameter string: [empty]
 - Icon path and file name: [empty]
- Buttons: Continue..., Cancel, Help

Figure 83. Transaction Program Definition Window

3. Type the **Transaction program (TP) name** using the value of **LOCALTP** (COBVSDT)
4. Using the value of **TPPATH**, type the **OS/2 Program path and file name** as:
drive:\IBMCOBOL\BIN\EQACEL62.EXE
5. Select the **Continue** push button. The **Additional TP Parameters** window appears, as shown in Figure 84.



The dialog box is titled "Additional TP Parameters". It contains the following options:

- Presentation type**
 - Presentation Manager
 - VIO-windowable
 - Full screen
 - Background
- Operation type**
 - Queued, Attach Manager started
 - Queued, operator started
 - Queued, operator preloaded
 - Non-queued, Attach Manager started
- Buttons: OK, Cancel, Help

Figure 84. Additional TP Parameters Window

6. For the **Presentation type**, select **Presentation Manager**.
7. For the **Operation type**, select **Non-queued, Attach Manager started**.

Verifying Your Configuration

8. Select the **OK** push button.
9. Select the **Close** push button on the **SNA Features List** window.
10. The **Communications Manager Profile List** window appears, as shown in Figure 73 on page 159

Verifying Your Configuration

1. Select **Close**.
2. The **Communications Manager Profile List** window appears, as shown in Figure 73 on page 159.
3. Select the **Close** push button and the **Communications Manager Configuration Definition - CVSCFG** window appears, as shown in Figure 85.

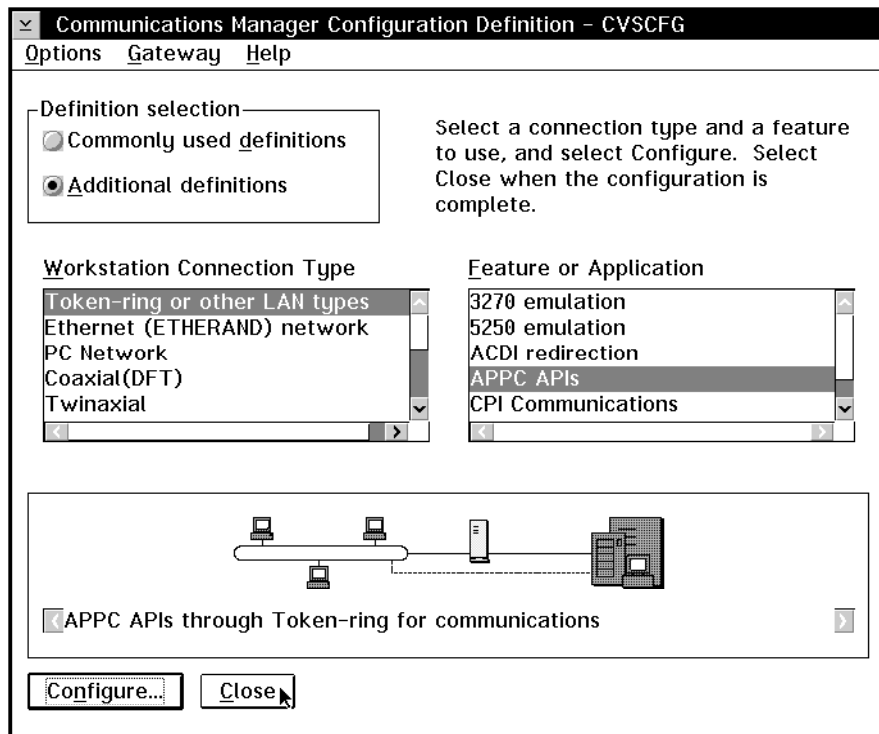


Figure 85. Communications Manager Configuration Definition - CVSCFG

4. Select the **Close** push button, and the **Communication Manager - Checking Values** window appears, as shown in Figure 86 on page 170. A progress indicator will appear on top of this window to indicate that CM/2 is verifying your APPC configuration.

Configuring for APPC Communications at MVS

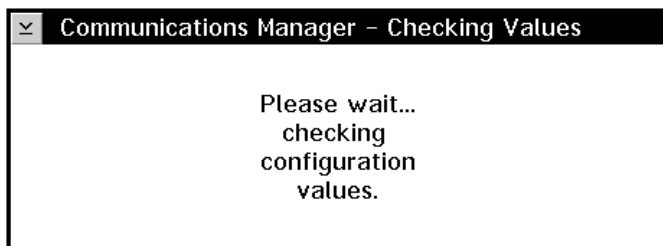


Figure 86. Communications Manager - Checking Values Window

If there are any errors in the APPC profile, you can identify them by examining the First Failure Support Technology/2 log within the FFST/2 folder on your OS/2 desktop. Make whatever corrections are necessary and repeat the steps above to verify your configuration. Your configuration can also be verified by issuing the following on an OS/2 command line within the CMLIB directory:

```
CMVERIFY CVSCFG
```

5. After your configuration has been verified, stop and restart CM/2 with the newly modified configuration file to activate your APPC configuration.

Configuring for APPC Communications at MVS

This section describes configuration for APPC at MVS. This configuration falls into two broad categories:

1. Configuring the APPC/MVS subsystem requirements (see *MVS/ESA Planning: APPC Management*)
2. Configuring VTAM, and possibly NCP, for APPC communications based on how workstations are connected (see *VTAM Resource Definition Reference* and *NCP Resource Definition Reference*)

Configuring the APPC/MVS subsystem is essentially independent of the type of network within which the workstations reside, and this section defines such configuring as it relates to VisualAge COBOL. However, VTAM and NCP definitions vary depending on network characteristics and the manner in which the workstations are connected.

Rather than attempt to address VTAM and/or NCP definitions for the many connections that are supported, we are restricting our description to APPC related definitions for a 3745 attached token-ring LAN.

Prerequisites

Software and hardware prerequisites for APPC/MVS include:

- MVS/ESA V4.2 or later
- VTAM V3.4 or later
- NCP V5.2 or later

APPC/MVS Configuration Variables

Terminology

Below is a table of standard APPC terms and the equivalent APPC/MVS terms. Some terms have no APPC/MVS equivalent, but instead are set in VTAM or NCP.

<i>Standard Term</i>	<i>APPC/MVS Term</i>
Network Name	NETID
LU Name	name supplied for APPL definition in APPL major node
Partner LU Name	name supplied for LU definition in major nodes defining each workstation
Local LAN Address	LOCADD (in NCP gen)
Adjacent LAN Address	DIALNO (on PATH definition in switched major node)

APPC/MVS Configuration Variables

Use the following table to create a worksheet that defines the listed symbols. These symbols and their corresponding values are used to configure APPC/MVS for communication with CM/2 at the workstation. Note that the worksheet consists of two parts:

1. Symbols that must be matched with the workstation (the partner platform) are under the heading **NETWORK VALUES**.
2. Symbols local to MVS are under the heading **LOCAL VALUES**.

<i>Table 4 (Page 1 of 4). Values Required to Configure APPC/MVS</i>		
<i>Symbol</i>	<i>How to determine the value for this symbol</i>	<i>Fill in your value here</i>
NETWORK VALUES		
LUNAME	<p>This symbol is for the APPC/MVS LU of the MVS system you are configuring. See "Define the APPC/MVS LU in VTAM" on page 175 and the bibliography for additional information on creating this LU. For MVS/ESA 4.2, use the base APPC/LU. See <i>MVS/ESA Planning: APPC Management</i>.</p> <p>This value should be consistent with PARTNERLU as defined at all participating workstations. See Table 3 on page 151 for more information about PARTNERLU at the workstation.</p>	
NETNAME	<p>This symbol is for the network name.</p> <p>For systems with VTAM already configured, this value is found in the VTAMLST start options (ATCSTRxx) member. The option name is NETID.</p> <p>This value should be consistent with PARTNERNET, as defined at all participating workstations within this network. See Table 3 on page 151 for more information about PARTNERNET at the workstation.</p>	

APPC/MVS Configuration Variables

<i>Table 4 (Page 2 of 4). Values Required to Configure APPC/MVS</i>		
Symbol	How to determine the value for this symbol	Fill in your value here
PARTNERLU	This symbol is for an associated workstation LU name. There should be a unique LU name for each participating workstation. This value should be consistent with the definition of LUNAME at the associated workstation. See Table 3 on page 151 for more information about LUNAME at the workstation.	
IDBLK	This symbol defines the first three hexadecimal digits of the NODEID for each participating workstation. Its value is normally 05D for a workstation. This value is defined in VTAM and its value must match the value defined in NODEID at the workstation. See Table 3 on page 151 for more information about NODEID at the workstation.	
IDNUM	This symbol defines the last five hexadecimal digits of the NODEID for each participating workstation. A unique IDNUM is defined in VTAM for each workstation. The last five hexadecimal digits of NODEID defined at the corresponding workstation must match IDNUM . See Table 3 on page 151 for more information about NODEID at the workstation.	
REMOTETP	<p>This is the symbol for the transaction program that resides on the workstation. It is transmitted across the network from APPC/MVS, and initiates a session with the Remote PWS Debug Tool server at the workstation. This value should be consistent with LOCALTP, as defined at all participating workstations. See Table 3 on page 151 for more information about LOCALTP at the workstation.</p> <p>This symbol will be used in defining APPC/MVS CPI-C side information entries for Remote PWS Debug Tool sessions.</p> <p>This symbol should have a value of COBVSDT.</p>	COBVSDT
LOCALTP	<p>This symbol is for the transaction program that resides on MVS. It is transmitted across the network from the workstation, and initiates a session with the APPC/MVS server.</p> <p>This value should be consistent with REMOTETP, as defined at all participating workstations. See Table 3 on page 151 for more information about REMOTETP at the workstation.</p> <p>Use the value COBOLVS_MVSLU62_EC_SERVER as the value of LOCALTP.</p>	
LOCALCP	<p>This symbol is for VTAM's System Services Control Point (SSCP) name. It is the value of SSCPNAME in the VTAM start options.</p> <p>This value should be consistent with PARTNERCP, as defined at all participating workstations.</p>	
LOCAL VALUES		

APPC/MVS Configuration Variables

Table 4 (Page 3 of 4). Values Required to Configure APPC/MVS

Symbol	How to determine the value for this symbol	Fill in your value here
NCPLANADDR	This symbol is for the local LAN address coded in NCP on a LINE macro using the LOCADD parameter. For our example, the workstation is configured through the 3745, but not through a different APPN network node. Therefore, this value should be consistent with LANADDRESS , as defined at all participating workstations on this token-ring. See Table 3 on page 151 for more information about LANADDRESS at the workstation.	
MODETAB	Name of a logon mode table in VTAM that contains the APPC mode entries.	
MODE	Name of the logon mode table entry within the table that is defined by MODETAB and used by VisualAge COBOL. Its recommended value should be #INTER . This value must match the value of MODE at the workstation. See Table 3 on page 151 for more information about MODE at the workstation.	#INTER
SIFILE	This symbol is for the VSAM data set where the CPI-C side information profile will be permanently located. This value is normally SYS1.APPCSI.	SYS1.APPCSI
SYMDEST	This is the symbol for the value representing a CPI-C side information symbolic destination name. Each participating workstation will have a unique SYMDEST value which points to an entry in a VSAM dataset. This entry identifies the corresponding workstation and is used by APPC/MVS during initialization of a Remote PWS Debug Tool session. The value of SYMDEST points to the following information for a workstation: <ul style="list-style-type: none"> • The value of REMOTETP (COBVSDT) • The value of PARTNERLU (the workstation LU name) • The value of MODE (#INTER) 	
TPFILE	This symbol is for the VSAM data set where the TP profiles will be permanently located. This value is normally SYS1.APPCTP.	SYS1.APPCTP
VOL	This is the name of a volume on which VSAM datasets for APPC/MVS are created.	
JES	The actual JES subsystem type (JES2 or JES3) should be substituted for this.	
CLIST.DATASET	The CLIST dataset in which you put the EVFSTR62 REXX exec. This is the data set for SEQACLIS DD of the Full Function feature of COBOL for MVS & VM Release 2 product (called SAA AD/CYCLE COBOL/370 for Release 1). If you are not using the Remote Edit/Compile component, you do not need a value for this symbol.	

APPC/MVS Definitions

Symbol	How to determine the value for this symbol	Fill in your value here
APPC.LOADLIB	The load library dataset in which you put the EVFM62IN program. This module is normally in the data set for SEQAMOD DD of the COBOL for MVS & VM product. If you are not using the Remote Edit/Compile component, you do not need a value for this symbol.	

APPC/MVS Configuration Overview

Configuring APPC/MVS requires the following:

- Add two members to SYS1.PARMLIB to define parameters for the APPC and ASCH subsystems, respectively.
- Define a TP profile and store it in the appropriate VSAM dataset
- Define a CPI-C side information entry for each participating workstation and store it in the appropriate VSAM dataset.
- Define to VTAM an application major node for the APPC/MVS LU
- Add APPC modes to the VTAM logmode table.

APPC configuration for the 3745 attached LAN in VTAM requires the following:

- Define a switched major node for the LAN
- Identify PU definitions for each workstation
- Identify a Type 2.1 (APPC) LU definition for each PU (or workstation)

APPC configuration for the 3745 attached LAN in NCP requires the following:

- Define the token-ring adapter in NCP
- Define APPC session limits in NCP

APPC/MVS Definitions

Refer to your worksheet and replace the highlighted symbols in APPC/MVS definitions with the corresponding values defined in the worksheet.

1. Create Parmlib members

Place two members (ASCHPMxx and APPCPMxx) in SYS1.PARMLIB. These two members have startup parameters for the two system components that make up APPC/MVS:

- APPC - this component communicates with VTAM on behalf of APPC applications (see Figure 87 on page 175)
- ASCH - this scheduling component handles incoming requests for local transaction programs (see Figure 88 on page 175)

APPC/MVS Definitions

```
/* APPCPM00 */
LUADD ACBNAME(LUNAME) /* Add LU LUNAME to the */
/* APPC/MVS configuration */
SCHED(ASCH) /* Specify that the APPC/MVS */
/* transaction scheduler is associated */
/* with this LU name */
BASE /* Designate this LU as the base LU */
TPDATA(TPFILE) /* Specify that VSAM data set */
/* TPFILE is the permanent */
/* repository for the TP profiles */
/* for this LU */
TPLEVEL(USER) /* Specify the search order for TP */
/* profiles as : */
/* 1. TP profiles associated with */
/* a specific user */
/* 2. TP profiles associated with */
/* a group of users */
/* 3. TP profiles associated with */
/* all users of the LU name */
SIDEINFO DATASET(SIFILE) /* Specify that VSAM data set */
/* SIFILE is the permanent */
/* repository for the side */
/* information */
```

Figure 87. SYS1.PARMLIB(APPCPM00) - APPCPM00.LIB

```
CLASSADD CLASSNAME(DEFAULT) /* Specify the name of the class to be */
/* added */
MAX(16) /* Specify that the maximum number */
/* of transaction initiators allowed */
/* for this class is 10 */
MIN(2) /* Specify that the minimum number */
/* of transaction initiators to be */
/* brought up for this class is 2 */
RESPGOAL(.02) /* Specify that the response time */
/* goal for transaction programs */
/* executing within this class is 0.02 */
/* seconds */
MSGLIMIT(500) /* Specify that the maximum size of */
/* the job logs for TPs is 500 */
/* messages */
OPTIONS DEFAULT(DEFAULT) /* Specify the default class */
SUBSYS(JES) /* Specify the name of a subsystem */
TPDEFAULT REGION(4M) /* Change the region size to 4M */
OUTCLASS(A) /* Change the output class to A */
```

Figure 88. SYS1.PARMLIB(ASCHPM00) - ASCHPM00.LIB

2. Define the APPC/MVS LU in VTAM

APPC/MVS Definitions

Refer to your worksheet and replace highlighted symbols in the VTAM APPL with corresponding values defined in the worksheet.

Place the following in a VTAMLST library (usually SYS1.VTAMLST):

```
COBOLVS VBUILD TYPE=APPL
LUNAME  APPL ACBNAME=LUNAME,APPC=YES,AUTOSES=10,DMINWNL=16,      *
        DMINWNR=16,DDRAINL=NALLOW,DRESPL=NALLOW,DSESLIM=64,EAS=64, *
        MODETAB=MODETAB,SECACPT=CONV,VPACING=2,VERIFY=NONE,      *
        SRBEXIT=YES,DLOGMOD=MODE
```

Figure 89. VTAM Definition of APPC LU

Note: The above definitions are recommended values. If the LU described by this APPL is shared with other APPC/MVS applications, the parameters might need to be altered. If you are running MVS/ESA 4.2, VisualAge COBOL requires the base LU and the resources defined by this APPL. If you are running MVS/ESA 4.3 or later, you can create an APPL to define an LU specifically for VisualAge COBOL. See *MVS/ESA Planning: APPC Management* for a description of base LUs.

3. Add APPC Modes to the Logmode Table

Below are the recommended APPC modes. Add them to your logmode table, compile, and link-edit the member into a VTAM library. Also include them in the default logmode table (ISTINCLM), if they are not already present, so that dynamically created LUs can use them.

For performance reasons, knowledgeable users can choose to use their own mode entry (as opposed to **#INTER**) that is based on their network characteristics. If a different mode is used, it must have a corresponding definition at each configured workstation.

Configuring to Run as a Server

```
*****
*      LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING      *
*      AS LU 6.2 DEVICES                                          *
*****
SNASVCMG MODEENT LOGMODE=SNASVCMG,FMPROF=X'13',TSPROF=X'07',
                PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',
                RUSIZES=X'8686',ENCR=B'0000',SSNDPAC=7,
                PSERVIC=X'0602000000000000000000300',
                SRCVPAC=7,PSNDPAC=7,TYPE=0
                TITLE '#INTER'
*****
*      LOGMODE TABLE FOR INTERACTIVE SESSIONS ON RESOURCES      *
*      CAPABLE OF ACTING AS LU 6.2 DEVICES                      *
*****
#INTER  MODEENT LOGMODE=#INTER,
                ENCR=B'0000',SSNDPAC=7,
                SRCVPAC=7,PSNDPAC=7,RUSIZES=X'F7F7'
                TITLE 'CPSVCMG'
*****
*      LOGMODE TABLE FOR CP-CP SESSIONS ON RESOURCES CAPABLE   *
*      OF ACTING AS LU 6.2 DEVICES                               *
*****
CPSVCMG MODEENT LOGMODE=CPSCVMG,
                RUSIZES=X'8686',ENCR=B'0000',
                SSNDPAC=7,SRCVPAC=7,PSNDPAC=7
                TITLE 'QPCSUPP'
*****
*      LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING      *
*      AS LU 6.2 DEVICES                                          *
*      REQUIRED FOR LU MANAGEMENT                                  *
*****
QPCSUPP MODEENT LOGMODE=QPCSUPP,FMPROF=X'13',TSPROF=X'07',
                PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',
                RUSIZES=X'8585',ENCR=B'0000',
                PSERVIC=X'0602000000000000000000300'
```

Figure 90. APPC Logon Mode Entries - APPCMODE.ASM

Configuring to Run as a Server

Defining Transaction Programs:

1. APPC/MVS uses a TP profile to schedule an inbound request for a Transaction Program. VisualAge COBOL requires such a TP profile for the server. If this dataset does not already exist, run the following job to create a VSAM dataset in which the TP profile can be stored.

Configuring to Run as a Server

```
//APPC001 JOB CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
/*JOBPARM LINES=9999,TIME=1440
//TPSAMPLE EXEC PGM=IDCAMS
//VOL1 DD DISP=OLD,UNIT=3380,VOL=SER=VOL
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//AMSDUMP DD SYSOUT=*
//SYSIN DD *
        DEFINE CLUSTER (NAME(TPFILE) -
            VOLUMES (VOL1) -
            INDEXED REUSE -
            SHAREOPTIONS(3 3) -
            RECORDSIZE(3824 7024) -
            KEYS(112 0) -
            RECORDS(300 150)) -
            DATA -
            (NAME(TPFILE.DATA)) -
            INDEX -
            (NAME(TPFILE.INDEX))
/*
```

Figure 91. Sample JCL for Creating TP Profile VSAM Dataset

2. If you are not using the Remote Edit/Compile component, skip this step and go to step 3 on page 179

Note: All of the Clist data sets described in this section with the prefix IGY are shipped with the full function feature of IBM COBOL for MVS & VM Release 2.

The following job runs the APPC/MVS administrative utility to add a TP profile for COBOLVS_MVSLU62_EC_SERVER. It starts the TSO Terminal Monitor Program IKJEFT01, which in turn calls the EVFSTR62 REXX exec that is in the Debug Tool installation dataset **EQAW.V1R2M0.SEQACLIS** (assuming default names were used at installation). The **IGY.V1R2M0.VB.SIGYCLST** data set is a copy of the IBM-supplied **IGY.V1R2M0.SIGYCLST** data set with a record format of VB (variable blocked). This enables it to be concatenated with the data set on the next line known as **CLIST.DATASET**. The block size (BLKSIZE) for **IGY.V1R2M0.VB.SIGYCLST** must be at least as large as the block size for **CLIST.DATASET**. The logical record length (LRECL) should be the same for both data sets.

The IBM COBOL for MVS & VM sample SYSADATA exit, **IGYADXIT**, must be compiled and link-edited. This exit is written in COBOL and is located in the IBM-supplied **IGY.V1R2M0.SIGYSAMP** data set. We recommend that you link-edit **IGYADXIT** into the IBM-supplied **IGY.V1R2M0.SIGYCOMP** data set, which is the same data set that contains the compiler load module **IGYCRCTL**. More information on the sample exit **IGYADXIT** can be found in *IBM COBOL for MVS & VM Compiler and Run-Time Migration Guide*.

IGYADXIT requires IBM Language Environment for MVS & VM Release 5. If the Release 5 run-time data set **CEE.V1R5M0.SCEERUN** is not already in the MVS link list, it must be concatenated to the STEPLIB DD statement for the TP profile.

Configuring to Run as a Server

```
//TPADD0 JOB CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
//STEP EXEC PGM=ATBSDFMU
//SYSPRINT DD SYSOUT=*
//SYSSDLIB DD DSN=USER.APPCTP,DISP=SHR
//SYSSDOUT DD SYSOUT=*
//SYSIN DD DATA,DLM=XX
TPADD
  TPNAME(COBOLVS_MVSLU62_EC_SERVER)
  ACTIVE(YES)
  TPSCHED_DELIMITER(##)
  TAILOR_SYSOUT(NO)
  TAILOR_ACCOUNT(NO)
  CLASS(DEFAULT)
  TPSCHED_TYPE(STANDARD)
  JCL_DELIMITER(END_OF_JCL)
//COBOLVS JOB (NNNN,I111),MSGLEVEL=(1,1),MSGCLASS=A
//IKJACCNT EXEC PGM=IKJEFT01,
// PARM='EXEC ''CLIST.DATASET(EVFSTR62)'''
//STEPLIB DD DISP=SHR,DSN=APPC.LOADLIB
/* CLIST/EXECs data set for IBM Language products
//SYSPROC DD DISP=SHR,DSN=IGY.V1R2M0.VB.SIGYCLST IBM COBOL for MVS & VM
// DD DISP=SHR,DSN=CLIST.DATASET
//SYSTSPRT DD DSN=&&TSOUT,SPACE=(TRK,(1,1)),DCB=(RECFM=V,LRECL=252,BLKSIZE=256,BUFNO=1)
//SYSTSIN DD DUMMY
//SYSPRINT DD SYSOUT=*,FREE=CLOSE
END_OF_JCL
  KEEP_MESSAGE_LOG(ERROR)
  MESSAGE_DATA_SET(&SYSUID.CODEPROD.LOG)
  DATASET_STATUS(MOD)
##
XX
/*
```

Figure 92. Sample JCL to Add the TP Profile to the VSAM Dataset

3. The following job runs the APPC/MVS administrative utility to add a TP profile for DFM/MVS.

Note: This might already have been done when the *Data Facility Storage Management Subsystem/MVS (DFSMS/MVS) Version 1 Release 2* product was installed.

```
//STEP EXEC PGM=ATBSDFMU
//SYSPRINT DD SYSOUT=*
//SYSSDOUT DD SYSOUT=*
//SYSSDLIB DD DSN=SYS1.APPCTP,DISP=SHR
//SYSIN DD DATA,DLM=XX
TPADD
  TPNAME(~X'07'001)
  ACTIVE(YES)
  TPSCHED_DELIMITER(##)
  CLASS(DEFAULT)
  JCL_DELIMITER(ENDJCL)
//GDEDFM JOB MSGCLASS=H,MSGLEVER=(1,1),CLASS=A
//GDEDFM EXEC PGM=GDEISASB
ENDJCL
##
XX
/*
```

Figure 93. Sample JCL to Add the DFM/MVS TP Profile to the VSAM Dataset

Defining the 3745 Attached LAN to VTAM

Configuring for Remote Debug Tool to Run as a Client

Defining CPI-C side information:

1. Create VSAM Dataset for CPI-C Side Information.

The following example job creates the VSAM dataset for CPI-C Side Information. It is necessary for subsequently defining side information entries allowing APPC/MVS to allocate sessions to an associated workstation. This dataset is defined to APPC/MVS in the SYS1.PARMLIB member APPCPMxx. In the subsequent example, side information entries will be defined in the dataset.

```
//APPC002 JOB CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
/*JOBPARM LINES=9999,TIME=1440
//SISAMPLE EXEC PGM=IDCAMS
//TSSC01 DD DISP=OLD,UNIT=3380,VOL=SER=VOL
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//AMSDUMP DD SYSOUT=*
//SYSIN DD *
        DEFINE CLUSTER (NAME(SIFILE) - /* side info data set name = SYS1.APPCSI */
        VOLUME(VOL) - /* the volume where the VSAM datasets are defined */
        INDEXED REUSE -
        SHAREOPTIONS(3 3) -
        RECORDSIZE(248 248) -
        KEYS(112 0) -
        RECORDS(50 25)) -
        DATA -
        (NAME(SIFILE.DATA)) - /* side info data set name = SYS1.APPCSI */
        INDEX -
        (NAME(SIFILE.INDEX))
```

Figure 94. Sample JCL for Side Information VSAM Dataset

2. Side information entries are defined in the above dataset by the sample JCL file ATBSIVSM.JCL.

```
//SIADD0 JOB CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
/******
//STEP EXEC PGM=ATBSDFMU
//SYSPRINT DD SYSOUT=*
//SYSSDLIB DD DSN=SIFILE,DISP=SHR
//SYSSDOUT DD SYSOUT=*
//SYSIN DD *
        SIADD
        DESTNAME(SYMDEST)
        TPNAM(TPNAME)
        MODNAME(MODE)
        PARTNER_LU(PARTNERLU)
/*
```

Figure 95. Sample JCL for Adding Side Information

Defining the 3745 Attached LAN to VTAM

To define the LAN and its workstations to VTAM:

- Step 1. Define a switched major node describing the LAN.

Defining the 3745 Attached LAN to VTAM

Step 2. Define a PU major node for a workstation.

Step 3. Define the workstation APPC/LU.

Repeat steps 2 and 3 for each workstation on the LAN. Vary PARTNERLU, IDBLK, and IDNUM on each workstation.

The example below illustrates key VTAM definitions relating to APPC configuration for your LAN and an associated workstation. These definitions are possibly incomplete and their actual values are for illustration purposes only. Your VTAM administrator and the bibliography should be consulted to create complete definitions.

```

WSN      VBUILD TYPE=SWNET,MAXGRP=2,MAXNO=2
PARTPU   PU      ADDR=04,                                X
          IDBLK=IDBLK,                                  X
          IDNUM=IDNUM,                                  X
          PUTYPE=2,                                     X
          MAXPATH=1,                                   X
PARTNER  LU      LOCADDR=0,MODETAB=MODETAB,DLOGMOD=MODE

```

Figure 96. Sample SYS1.VTAMLST(WSN)

PU Definition: Note that IDBLK and IDNUM are used to identify the workstation during VTAM's bind process. This value must be the same as the Node ID that is specified during CM/2 configuration. See Table 3 on page 151 for more information about **NODEID**.

As an alternative to IDBLK and IDNUM, CPNAME=PARTNERLU can be specified on the PU to identify the workstation during the bind process. In this case, **NODEID** does not have to be specified during CM/2 configuration.

PUTYPE=2 indicates that the workstation is capable of supporting independent LUs, which is a prerequisite for VisualAge COBOL.

For MODETAB and DLOGMOD, use the values defined on your worksheet. See *VTAM Resource Definition Reference* for additional values that you can use to define the workstation.

LU Definition: **PARTNERLU** is the LU name that defines the workstation to VTAM. This value must be the same as the LU name that is specified during CM/2 configuration. See Table 3 on page 151 for more information about **LUNAME** at the workstation.

LOCADDR=0 identifies this as an independent APPC LU for the workstation that is being defined by the PU statement.

While no LUs are illustrated for 3270 emulator sessions, it is likely that emulators will be used with VisualAge COBOL remote debugging and they will be present in each PU and LU group that defines a workstation. See *VTAM Resource Definition Reference* for additional information on how to define APPC and emulator LUs.

APPC/MVS System Commands

Defining the 3745 Attached LAN to NCP

The examples below illustrate sample NCP definitions relating to token ring and LAN APPC support. These definitions are possibly incomplete and their explicit values will vary for a given network. Therefore, these samples are for illustration purposes only. An NCP administrator should be consulted or see the references listed in the bibliography in order to create the appropriate definitions.

1. Define the Token-ring adapter in NCP

The following is a sample token-ring connection from NCP:

```
T030T2PG GROUP ECLTYPE=(PHYSICAL,ANY)
T030T2PL LINE ADDRESS=(1089,FULL),LOCADD=NCPLANADDR,PORTADD=2, X
RCVBUFC=4095,MAXTSL=2044,ADAPTER=TIC2,TRSPPEED=4
*
T030TRL0 GROUP ECLTYPE=LOGICAL,AUTOGEN=10,PHYPORT=2,CALL=INOUT
```

Figure 97. LAN definitions on NCP

2. Provide for the use of host (APPC/MVS and/or CICS) LUs on the token-ring connection

Once the token-ring is defined in the NCP, the only additional required parameter is NUMILU on the LUDRPOOL macro. An example is:

```
POOL1 LUDRPOOL NUMILU=100,NUMTYP1=20,NUMTYP2=20
```

In this example, the NCP can support up to 100 host (APPC/MVS or CICS) LUs that are connected via token-ring connections.

APPC/MVS System Commands

Commands to Start APPC/MVS:

```
START APPC,SUB=MSTR,APPC=xx
START ASCH,SUB=MSTR,ASCH=xx
```

The xx is the identifier of the parmlib member. The default is 00.

Before you can start DFM/MVS, you must update the SYS1.PARMLIB member DFM00 and verify that the PROCLIB member DFM exists. Figure 98 shows the DFM00 parameters member, and Figure 99 on page 183 shows the DFM procedure.

```
DFM LOCK_WAIT_INTV(20)
MAX_CONV_LOCK(5)
LOCK_RETRY(3)
CCSID(0)
CLOSE_CHECK_INTV(0)
DEFER_CLOSE_TIME(0)
MAX_AGENT_TSKS(5)
STREAM_LRECL(8196)
SEND_BUFFER_THRESHOLD(100)
```

Figure 98. DFM/MVS: Parameters Member in SYS1.PARMLIB

Configuring CICS for APPC Communications

```
//DFM   PROC PARM=NORMAL
//*****
//*   ADSTAR DISTRIBUTED FILE MANAGER ADDRESS SPACE   *
//*****
//       EXEC PGM=GDEISBOT,PARM='&PARMS',REGION=32M,TIME=1440
//IEFPARM DD DSN=SYS1.PARMLIB,DISP=SHR
//SYSPRINT DD SYSOUT=H
```

Figure 99. DFM/MVS: Startup Procedure

Before you start DFM/MVS, make sure that the APPC/MVS and PWS LUs are activated in ACF/VTAM and that APPC/MVS is started.

The following command starts DFM/MVS:

```
START DFM,SUB=MSTR
```

Two address spaces are created: DFM and DFMCAS.

Monitoring APPC: Once APPC/MVS is activated, there are several commands that can be used to monitor APPC/MVS operation. A subset of these commands is shown below:

```
DISPLAY APPC,TP,ALL
DISPLAY APPC,LU,ALL
DISPLAY ASCH,ALL
```

These commands provide information on APPC/MVS TPs, LUs, and scheduler operations, respectively.

Commands to Terminate APPC/MVS:

```
C APPC
C ASCH
```

For more information on APPC/MVS system commands, see *MVS/ESA Planning: APPC Management*.

Configuring CICS for APPC Communications

This section describes configuring CICS for Remote PWS Debug Tool APPC sessions. Assuming that administrators will configure Remote PWS Debug Tool APPC sessions at both APPC/MVS and CICS, some important differences should be noted. CICS interfaces directly with VTAM (that is, it runs as a VTAM application.) For that reason, the APPC/MVS subsystem is not used for CICS APPC sessions. Instead, APPC/MVS-like functions are performed by the CICS region. When configuring for Remote PWS Debug Tool, this is evident while defining CPI-C side information, which is done using CICS system transactions as opposed to using the APPC/MVS facilities that were defined previously. Also, since CICS is a VTAM application, it is assumed that you already have an APPL major node which serves as the LU for APPC sessions. That is, CICS uses its own APPL and not the one defined for APPC/MVS.

Terminology

Configuring CICS for Remote PWS Debug Tool sessions generally falls into two broad categories:

1. Configuring CICS for Remote PWS Debug Tool APPC session support
2. Configuring VTAM, and possibly NCP, on how the workstations are connected.
See the *VTAM Resource Definition* and the *NCP Resource Definition* manuals for more information.

The CICS configuration that is described is essentially independent of the type of network within which the workstations reside. However, network definitions in VTAM and NCP will vary depending on how partner workstations are connected. As was the case for MVS, our description is limited to a token ring LAN attached to the host via a 3745 communications controller (see “Defining the 3745 Attached LAN to VTAM” on page 180 and “Defining the 3745 Attached LAN to NCP” on page 182.) For further assistance in defining your network, contact your IBM representative or refer to the following publications:

These publications provide detailed information on the configuration process for CICS (V3.3 and V4.1) and VTAM (V3.4.1).

- *CICS/ESA: Resource Definition Guide* (SC33-1166)
- *CICS/ESA: System Definition Guide* (SC33-0664)
- *VTAM: Resource Definition Reference* (SC33-0666)
- *VTAM: Network Implementation Guide* (SC31-6434)
- *NCP: Resource Definition Reference* (SC30-3448)

Prerequisites

The following software levels are required for the definitions in this chapter:

- CICS/ESA 3.3 or later
- MVS/ESA 4.2 or later
- VTAM 3.4 or later
- NCP 5.2 or later

Terminology

Below is a table of standard APPC and equivalent CICS terms. Using standard terms makes configuring unlike platforms such as CICS and CM/2 for communications easier so standard terms are used whenever possible. Some terms have no CICS equivalent but instead are set in VTAM or NCP.

Standard Term	CICS Term
Link Name	PU name (VTAM)
Network Name	Network, NETID (see note)
LU Name	APPLID, ACBNAME
Partner LU Name	Netname
LAN Address	LOCADD (NCP)
Adjacent LAN Address	DIALNO (VTAM)

CICS Configuration Variables

Note: CICS uses a network name only when defining CPI-C side information in a PARTNER definition.

CICS Configuration Variables

Use the following table to create a worksheet that defines the listed symbols. These symbols and their corresponding values will be used to configure CICS for communication with CM/2 at the workstation. Note that the worksheet consists of two parts:

1. Symbols that must be matched with a partner platform (the workstation.) The symbols are under the heading **NETWORK VALUES**.
2. Symbols that are local to CICS. These symbols are under the heading **LOCAL VALUES**.

<i>Symbol</i>	<i>How to determine the value for this symbol</i>	<i>Fill in your value here</i>
NETWORK VALUES		
PARTNERLU	<p>This is the symbol for an associated workstation name. There should be a unique LU name for each participating workstation. This value should be consistent with LUNAME at the workstation. See Table 3 on page 151 for more information about LUNAME at the workstation.</p> <p>When CPI-C side information is created, a value for PARTNERLU must be supplied within a side information entry for each participating workstation.</p>	
PARTNERNET	<p>This is the symbol for the workstation's network name. We are assuming that CICS and the workstations are in the same network. In this case, the value can be obtained from the VTAM start option (NETID).</p> <p>This value should match NETWORK as defined to CM/2 at each participating workstation. See page 152 for more information about NETWORK at the workstation.</p>	
TPNAME	<p>This is the symbol for the Remote PWS Debug Tool transaction program residing at the workstation. It is transmitted across the network from CICS and it initiates a session with the Debug Tool server at the workstation. This value should be consistent with LOCALTP, as defined at all participating workstations. See Table 3 on page 151 for more information about LOCALTP at the workstation.</p> <p>For Remote PWS Debug Tool, specify the value COBVSDT.</p>	COBVSDT
MODETABLE	<p>The binary table containing mode names that VTAM searches, usually a member of SYS1.VTAMLIB. Verify that the table defined in VTAM for the CICS LU has the required mode names (#INTER and SNASVCMG) for Remote PWS Debug Tool.</p>	
MODE	<p>Name of the logon mode table entry that VTAM uses for Remote PWS Debug Tool VTAM sessions. #INTER is recommended. See Figure 90 on page 177 for a list and description of mode entries that may be used by both MVS/APPC and CICS.</p>	#INTER
LOCAL VALUES		
GROUP	<p>A name for the collection of CICS definitions. CICS reserves DFH for its own use, so choose a name like COBOLVS.</p>	

Symbol	How to determine the value for this symbol	Fill in your value here
CON	A <i>connection</i> name used as a nickname for the workstation PARTNERLU . Such a definition is required for each PARTNERLU . Several other parameters are tied to the connection name. See Table 3 on page 151 for more information about PARTNERLU at the workstation.	
SESSION	A name that uniquely identifies a session for each participating workstation and defines the MODE for that session. See “Defining SESSIONS” on page 188.	
SYMDEST	This is the symbol for a value that represents a CPI-C side information symbolic destination name. Each participating workstation will have a unique SYMDEST value which identifies the workstation, mode, and transaction profile name at session initialization. In CICS, a side information entry is defined by a PARTNER definition.	
PROFILE	A name defining partner characteristics for this group in CPI-C side information. See “Defining Side Information” on page 189.	

Overview of CICS APPC Configuration

Configuring CICS for Remote PWS Debug Tool APPC sessions requires:

1. Configuring CICS to run as a client
 - a. Defining CPI-C side information
 - b. Performing other client-related configuration activities
2. Defining a Link to the LAN

Note: The SIT parameter ISC=YES must be specified.

Configuring CICS to Run as a Client

The following provides instructions for using CICS Resource Definition Online (RDO) to create and modify CICS definitions. To start Resource Definition Online, you must be logged on to CICS and your user-ID must be authorized to run the CEDA transaction.

You must define the following:

- a CONNECTION object (with the partner LU name)
- a SESSIONS object (with the mode name for the session)
- a PARTNER
- a PROFILE

In CICS, *objects* have properties such as authorization lists (which user-ids can use the object), resources required (memory, buffers), etc. CICS manages all these to provide low response time.

Defining CONNECTIONS

Note: To distinguish CICS objects, such as a CONNECTION, from more generic usages, the CICS object names will be in upper case.

To create and modify definitions in CICS start the CEDA transaction. All definitions in CICS are kept in *groups*. Most object identifiers must be globally unique, that is, cannot be used in any other group. SESSIONS objects are an exception to this rule. In the following sections all the definitions are in the same group.

Defining CONNECTIONS

A connection must be defined for each workstation participating in Remote PWS Debug Tool APPC sessions with CICS.

To define a connection in CICS, type the following at a clear screen:

```
ceda define connection(CON) group(GROUP)
```

CICS folds all characters to upper-case, so you don't have to worry about typing lower-case characters. If the group does not exist, CICS will create it for you automatically. Figure 100 on page 188 shows the 3270 screen that results from this command. CICS fills in the group name and the CONNECTION identifier from the values you typed. On this screen and all other CICS screens, if a value must be selected from a list, then the part of the list value that is upper-case serves as a short-name. For example, the ACcessmethod parameter has options of Vtam, IRc, INdirect, and Xm. To select Vtam, typing "V" is the same as typing "VTAM."

Defining SESSIONS

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0330
CEDA DEFINE
Connection   : CON                               # note 1
Group        : GROUP
Description  ==> PARTNER LU
CONNECTION IDENTIFIERS
Netname      ==> PARTNERLU                       # note 2
INDsys       ==>
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME   ==>
CONNECTION PROPERTIES
ACessmethod ==> Vtam                             Vtam | IRc | INdirect | Xm
Protocol     ==> Appc                             Appc | Lu61           # note 3
SInglesess   ==> No                               No | Yes             # note 4
DATastream   ==> User                             User | 3270 | SCs | STRfield | Lms
RECORDformat ==> U                               U | Vb
OPERATIONAL PROPERTIES
+ AUTOconnect ==> Yes                             No | Yes | All      # note 5
INSERVICE    ==> Yes                             Yes | No
SECURITY
SEcurityname ==>
ATTachsec    ==> Local                           Local | Identify | Verify | Persistent
                                                    | Mixidp
BINDPassword ==>                                PASSWORD NOT SPECIFIED
BINDSecurity ==> No                             No | Yes

APPLID=LOCALLU

```

Figure 100. CEDA DEFINE CONNECTION(CON) GROUP(GROUP)

Required parameters:

1. *Connection* is a 1-4 character nickname for this partner LU.
2. *Netname* is the **PARTNERLU** name. This must match the **LUNAME** defined for the corresponding workstation.
3. *Protocol* must be entered and must be APPC.
4. *SInglesess* must be set to NO. The example definitions in this guide all require parallel sessions.
5. The “+” sign indicates the bottom of the first screen displayed by CEDA. The function key settings are displayed at the bottom of the screen (but not shown here).

Here the default, “Local,” means that any transaction that can be run by the default user-ID, CICSUSER, can be run by this partner LU.

Defining SESSIONS

Define the mode name and session characteristics for each **PARTNERLU** (or workstation) with a sessions definition that can be initiated by the following command:

```
ceda define sessions(SESSION) group(GROUP)
```

Figure 101 on page 189 shows the 3270 screen that you should get as a result.

Defining Side Information

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0330
CEDA DEFINE
Sessions      : SESSION                          # note 1
Group         : GROUP
Description   ==> MODE SESSIONS FOR PARTNER LU
SESSION IDENTIFIERS
Connection    ==> CON                              # note 2
SESSName     ==>
NETnameq     ==>
MODename     ==> MODE                              # note 3
SESSION PROPERTIES
Protocol      ==> Appc                               Appc | Lu61
MAXimum      ==> 008 , 004                          0-999    # note 4
RECEIVEPfx   ==>
RECEIVECount ==> 1-999
SENDPfx      ==>
SENDCount    ==> 1-999
SENDSize     ==> 04096                               1-30720
+ RECEIVESize ==> 04096                               1-30720

APPLID=TOCICS3
```

Figure 101. CEDA DEFINE SESSIONS(SESSION) GROUP(GROUP)

Required parameters:

1. *SESSIONS* is the CICS name for this session set. The name must be unique within this group.
2. *Connection* identifies the CICS connection in this group (in other words, the partner LU) associated with this mode name.
3. *Modename* specifies the mode for this group of sessions.
4. *Maximum* specifies the number of sessions and contention winners for this node name.

Defining Side Information

Side information must be defined for each workstation participating in Remote PWS Debug Tool APPC sessions with CICS.

To define CPI-C side information in CICS, you must create a PARTNER definition and a PROFILE definition.

To create a PARTNER definition, type the following command in CICS:

```
ceda define partner(SYMDEST) group(GROUP)
```

The 3270 screen shown in Figure 102 on page 190 should appear. Type in the values for Partner, Netname, Network, and TP name from the worksheet in “CICS Configuration Variables” on page 185.

Defining Side Information

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0330
CEDA DEFINE
PARTNER      : SYMDEST                             # note 1
Group       : GROUP
Description  ==> CPI-C SIDE INFO
REMOTE LU NAME
NETName     ==> PARTNERLU                          # note 2
NETWork     ==> PARTNERNET                         # note 2
SESSION PROPERTIES
Profile     ==> PROFILE                            # note 3
REMOTE TP NAME
Tpname      ==> TPNAME                             # note 4
           ==>
Xtpname     ==>
           ==>
           ==>

APPLID=TOCICS3
```

Figure 102. CEDA DEFINE PARTNER

Required parameters:

1. *Partner* is the name for this resource in CICS.
2. *Netname* is the partner's LU name. *Network* is the partner's network name.
3. *Profile* is the set of characteristics this partner has. The most important of these is the modename.
4. *Tpname* is the initial setting for the remote TP name. The TP name must match a corresponding TP definition at the workstation.

To define a PROFILE to explicitly set the mode name for the side information, type the following command:

```
ceda define profile(PROFILE) group(GROUP)
```

The screen shown in Figure 103 on page 191 should appear. Type in the values for PROFILE and MODE from the worksheet in "CICS Configuration Variables" on page 185.

Installing the Definitions

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0330
CEDA DEFine
  PROFile      : PROFILE                            # note 1
  Group       : GROUP
  DEscription ==> PROFILE TO CHANGE MODE NAME
  Scrnsz     ==> Default      Default | Alternate
  Uctran     ==> No          No | Yes
  MOdename   ==> MODE                            # note 2
  PRIntercomp ==> No          No | Yes
JOURNALLING
  Journal    ==> No          No | 1-99
  MSGJrn1   ==> No          No | INPut | OutPut | INOut
PROTECTION
  MSGInteg  ==> No          No | Yes
  Onewte    ==> No          No | Yes
  PROtect   ==> No          No | Yes
  Chaincontrol ==> No       No | Yes
PROTOCOLS
+ DVsuprt   ==> All         All | Nonvtam | Vtam
APPLID=TOCICS3

```

Figure 103. CEDA DEFINE PROFILE(PROFILE) GROUP(GROUP)

Required parameters:

1. *Profile* is the name for this resource. This name must match the Profile parameter in the PARTNER definition.
2. *Mode name* identifies the mode for this side info entry. There must be a SESSIONS definition with the same modename. (Remember, this profile entry is optional if the application uses a default mode name, as APING does.) We recommend for Remote PWS Debug Tool that MODE be **#INTER**.

Installing the Definitions

Before **any** definitions can be used they must be added to the running CICS System Definition (CSD) using the INSTALL command. This is accomplished by typing:

```
ceda install group(GROUP)
```

This command will install all the objects in the group - CONNECTIONS, SESSIONS, etc. Some objects can be installed individually, too. If a resource is in use, or *In Service* in CICS terms, the installation will fail. If you get the INSTALLATION FAILED message, you can display the messages by pressing the PF9 key. However, this should not happen the first time.

If you need to acquire a CONNECTION, use the CEMT transaction.

Note: The above install command must be repeated after each cold start of the CICS region. If this results in a problem, use the following command instead:

```
ceda add group(GROUP) list(LIST)
```

Defining a Link to the LAN

These are identical when configuring for APPC/MVS. Check to see that such configuring has not already been done before proceeding further. If these definitions have not been performed, see “Defining the 3745 Attached LAN to VTAM” on page 180 or “Defining the 3745 Attached LAN to NCP” on page 182 for more information.

CEMT to Display/Modify Sessions

In CICS, the CEMT transaction allows you to display and modify the status of connections, netnames, and modenames (that is, all sessions with a particular mode name):

```
cent inq netname(PARTNERLU)
cent inq modename(CON)
```

You may need to manually start a session between CICS and the partner LU. To start a session with the partner session, at a CICS screen type this command:

```
cent inq con(CON)
```

You will get the screen shown in Figure 104.

```
INQ CON(CON)
STATUS: RESULTS - OVERTYPE TO MODIFY
Con(CON) Net(PARTNERLU)   Ins Rel          NORMAL
```

Figure 104. CICS Status for Connection CON

By the connection and netname for the partner LU you should see “Ins Acq.” This is CICS short-hand for “In-service, acquired.” If the status is “Ins Rel,” overtype the “Rel” with “Acq” and press the enter key. The resulting screen should now have “Ins Acq” as the status. If not, contact your VTAM systems programmer and make sure that the partner LU is active. You can also use the VTAM command to display activate the partner LU or switched major node.

When the connection is in service and acquired, you can start to use it for Remote PWS Debug Tool communications.

Appendix D. REXX Procedures: Compile and Link

This appendix describes the sample REXX procedures supplied with IBM COBOL for MVS & VM Release 1.2. to compile and link COBOL programs. All of these examples have been supplied within the data set, SIGYCLST on the IBM COBOL for MVS & VM product tape. These sample REXX procedures are used by the Remote Edit/Compile component which is provided only at beta level.

Note: All of the supplied REXX sample procedures need to be customized for your site or replaced by your own REXX procedures.

REXX Procedures for COBOL Programs

IGYFCIC	Translates and compiles a CICS COBOL program in the foreground
IGYFCL	Compiles and links a COBOL program in the foreground
IGYFC	Compiles a COBOL program in the foreground
IGYBDB2	Translates, compiles, and links a DB2 COBOL program in batch (DB2 precompile, compile)

REXX Procedures for DB2 Programs

IGYFBIND	Binds DB2 programs in the foreground
-----------------	--------------------------------------

Other REXX Procedures

IGYFINIT	Allocates private datasets used by sample REXX procedures
IGYFCIL	Link-edits CICS object(s) in the foreground for production
IGYFCILD	Link-edits CICS object(s) in the foreground for debug
IGYFL	Link-edits COBOL object(s) in the foreground
IGYFVALS	Sets installation environment variables used by the other REXX procedures

REXX Procedures

VisualAge COBOL Glossary

This glossary contains terms commonly used in VisualAge COBOL.

Action. In WorkFrame, a description of a tool or function that can be used to manipulate a project's parts, or build a project's target.

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 COBOL GUI Designer, a GUI project whose target is built as an EXE instead of a DLL.

APPC. Advanced program-to-program communication. Communications protocol between the workstation and the MVS host. SdU for remote edit and compile, including the debugger, uses the APPC communications protocol.

build. An action that invokes the WorkFrame Build tool to create the project's target. The Build tool manages the project's makefile, as well as build 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, WorkFrame's makefile creation utility.

child project. A project contained by another project. See also *nested project*.

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.

compile. To translate a source program into an executable program (an object program).

component. (1) A functional grouping of related files. (2) In the COBOL GUI Designer, a GUI project whose target is built as a DLL instead of an EXE.

copy file. A file with the extension CPY that acts as a copybook.

COBOL GUI Designer Master Project. A master project that provides all the default Tools setups and build mechanisms to create a COBOL GUI project. A

COBOL GUI project does not allow you to create non-GUI projects, although you can nest a non-GUI project within a COBOL GUI project.

COBOL Master Project. A master project that provides all the default Tools setups and build mechanisms to create a COBOL project. A COBOL project that inherits from the COBOL Master Project enables you to create GUI projects.

Common User Access architecture. Guidelines intended to help product designers and developers create an interface that users will find easy to learn and use.

CUA architecture. See *Common User Access architecture*.

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. 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.

environment variable. 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 any other environment variables that are defined using the OS/2 SET command, such as SYSLIB.

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.

file-scoped action. Distinguished from a project-scoped action in that it is invoked on files. Only file-scoped actions can participate in a project build.

Glossary

filter. In WorkFrame, the value of a type. The filter of a type can be expressed as a file mask, regular expression, a logical-OR, a logical-AND, or logical-NOT of a list of types, or a filter determined by a PAM.

Get file... Menu choice that enables you to import a file into the COBOL Editor.

GUI. Acronym for graphical user interface.

information area. A part 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. In Workframe and Data Assistant, the information area.

inheritance. In WorkFrame, refers to the mechanism in which the tools setup of a project is shared by another project.

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.

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

MakeMake. WorkFrame's makefile generation utility.

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

make utility. A tool that automates the process of updating project parts. The make utility compares the modification dates for one set of parts (the target parts) with those of another set of files (the dependent parts, such as source files). If any dependent parts have changed more recently than the target parts, the make utility runs a series of commands to bring the targets up-to-date.

master project. A project from which another project inherits its tools setup. Distinguished from *parent project*.

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

migrate. To move to a changed operating environment, usually a new release or version of a system.

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

nested project. A project that appears inside another project. Nesting expresses a dependency of the parent project on the child project's target. This dependency is managed by WorkFrame's Build utility.

object. A visual component of a user interface that a user can work with to perform a task. That you work with to perform a task. Text, graphics, and icons on the desktop are examples of objects.

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.

PAM. See *Project Access Method*.

parent project. A project that contains other projects.

part. (1) In WorkFrame, same as a *project part*. (2) In the COBOL GUI Designer, the graphic representations of GUI controls that you use to create a GUI, for example, a push button part.

Parts Catalog. In the COBOL GUI Designer, a notebook that contains all of the parts you can use to build a GUI.

Parts Palette. In the COBOL GUI Designer, a window that contains a commonly used set of parts you can use to build a GUI. You can customize the Parts Palette by including parts available in the Parts Catalog.

program. A syntactic unit that conforms to the rules of a particular programming language composed of declarations and statements or instructions needed to solve a certain function, task or problem. Synonymous with computer program.

project. The central WorkFrame model of the complete set of data and actions required to build a target, such as a dynamic link library (DLL) or other executable (EXE). A project consists of a set of *project parts* and a *Tools setup*.

Glossary

Project Access Method (PAM). A dynamic link library that contains a set of methods through which a simple abstraction of a file system or repository is provided to WorkFrame. PAMs enable a WorkFrame project to contain any kind of object that a PAM can support, for example a version of a file in a source control library, or another file system like MVS or AIX.

project hierarchy. A project tree that represents dependencies between projects. The WorkFrame 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 parts. The data objects that make up a WorkFrame project. As well as a source file, a project part may also be a transient object, such as a target or an intermediate object created during the life of the project. A project part may also be another project.

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

Project Smarts. A catalog that contains predefined projects.

SOM. See *System Object Model*.

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

source type. A source type appears in an action's list of source types. An action's list of source types specifies the kind of parts or files to which the action applies.

subprogram. 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.

status area. A part 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.

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.

target. In WorkFrame, a project's target is 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 target type appears in an action's list of target types. Target types 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 a template, you create a copy of the original object. The new object has the same settings and contents as the original template object.

Tools setup. A view of a project where you can see and manipulate the actions, types, and environment variables available to the project. From this view, you can add, delete, and change actions, types, and variables. You can also set the options for any action in this view.

type. In WorkFrame, describes a group of project files of parts in terms of an expression, such as file masks, regular expressions, or a list of other types, logical-OR'd.

type class. In WorkFrame, represents the method by which an object is determined to be a member of a type. "File mask" is an example of a type class. Membership to a "File mask" type is determined by matching the file mask filter to the object's name. Other examples of type classes are "Regular expression," and "PAM Name," where the named Project Access Method determines membership to a type.

thread. In the OS/2 operating system, the smallest unit of operation to be performed within a process.

Glossary

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.

view. In the OS/2 operating system, the appearance of the contents of an open object.

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*.

Comparison of Workstation and Mainframe Concepts

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 5. Comparison of workstation and mainframe concepts

Workstation Concept	Mainframe Concept
File : A unit of stored information for text, data, programs, etc.	Data set, member, or file : In MVS, a data set is the major unit of data storage and retrieval. A member is a partition of a partitioned data set.
Command file (.CMD) : a file containing OS/2 commands organized for sequential processing. In workstation programs, command files are much like JCL.	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 command file can also function similarly to a CLIST or an EXEC .	CLIST in MVS, EXEC in VM: 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.
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.	No equivalent term or concept on the mainframe.
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 or in a shared segment on VM.
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.
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.
OS/2 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 OS/2 operating system.	No exact equivalent. The closest equivalent on the mainframe are menu-driven tools such as ISPF or Office Vision.

Comparison of Workstation and Mainframe Concepts

Index

A

- adding
 - components 19, 22
 - GUI parts 94
- administrator tasks
 - preparing the server 5
- APPC configuration
 - CICS 183
 - CM/2 150
 - MVS 170
- APPC/MVS configuration
 - definitions 174
 - overview 174
 - prerequisites 170
 - running as a client (Remote PWS Debug Tool) 180
 - running as a server (E/C) 177
 - variables 171
- APPC/MVS, system commands to start 182
- application
 - creating, logic 97
 - example, GUI 89
 - packaging for distribution 113
- attributes
 - changing 95

B

- build
 - application 108
 - error messages 109

C

- choosing what to install 9
 - determining
 - required space 10
- CICS
 - configuration 183
 - configuration variables 185
 - defining a connection 187
 - defining a session 188
 - defining side information 189
 - installing the definitions 191
 - overview 186
 - prerequisites 184

- CICS (*continued*)
 - to run as a client 186
- CID support
 - return codes 32
- CM/2
 - configuring for APPC 150
- CM/2 APPC configuration
 - defining
 - connection 160
 - partner LU (E/C) 163
 - workstation to network 157
 - examples 154
 - Remote Debug Tool 167
 - required variables 151
 - selecting configuration file 155
 - to run as a client 165
 - verification 169
- COBOL Editor window 97
- COBOL GUI Designer
 - coding event logic 59
 - description 49
 - disk space 4
 - environment variables 14
 - saving a project 55
 - using 52
- COBOL program, REXX procedures for 193
- Code Assistant
 - description 69
 - using 98
- command-line options 31
- commands
 - DIAMOND 12
 - starting APPC/MVS 182
- communications
 - configuring 149
 - synchronous 150
- Communications Manager profile list, returning to 164
- compile and link
 - using REXX 193
- Compiler and Nonvisual tools
 - Compiler and Runtime 4
 - Editor 4
 - environment variables 13
 - Information 4
 - Interactive Debugger 4
 - SMARTdata UTILITIES 4

- Compiler and Runtime
 - disk space 4
 - environment variables 13
- Compiler Default Options Tool
 - DIAMOND command 12
 - invoking 12
 - window 11
- compiler options
 - Debug page 105, 106
 - link page 106, 107
 - notebook 104
 - Other page 105
 - setting 103
- components
 - adding 19
 - adding after initial installation 22
 - deleting 19, 20
 - disk space 4
 - reinstalling 19
 - requiring other components 9
 - selecting to install 9
- CompuServe 83
- CONFIG.ADD file 13
- CONFIG.SYS file
 - updating automatically 8, 22
 - updating yourself 13
- configuration
 - APPC/MVS 170
 - CICS 183
 - CM/2 APPC 150
 - variables
 - APPC/MVS 171
 - CM/2 151
- configuration variables
 - CICS 185
- configuring APPC communications 149—192
- consulting services
 - information 84
 - World Wide Web 84
- corrective service disk (CSD) 7
- Create New Project window 90, 93
- creating
 - application with a GUI 89
 - GUI project 93
 - new project 43, 90
 - subroutine logic 91
 - subroutine project 90
- CSD 7
- customizing
 - GUI parts 94

D

- Data Assistant
 - description 70
 - disk space 4
 - environment variables 14
 - tutorial 117
- Data Description and Conversion
 - disk space 4
 - environment variables 15
- DB2 program, REXX procedures for 193
- debugging
 - remote Debug Tool 167
 - setting option 106
 - setting options 105
 - using the debugger 110
- deleting components 19, 20
- determining
 - available disk space 10
 - required space 10
- development tools, Warp
 - disk space 4
- DIAMOND command 12
- disk space
 - COBOL GUI Designer 4
 - Compiler and Runtime 4
 - Data Assistant 4
 - Data Description and Conversion 4
 - determining
 - development tools, Warp 4
 - Editor 4
 - FAT partition 5
 - headers and libraries, Warp 4
 - Interactive Debugger 4
 - multimedia bitmaps, Warp 4
 - Performance Analyzer 4
 - Remote Edit/Compile 4
 - requirements 3
 - sample programs, Warp 4
 - SMARTdata UTILITIES 4
 - swapper 4
 - Transaction Assistant 4
 - WorkFrame 4
- distributing application to users 115
- documentation
 - ordering 81
 - printing 81
 - viewing 80

E

- editing
 - language sensitive 91
- Editor
 - creating a new file 45
 - description 69
 - disk space 4
 - environment variables 14
 - language sensitive editing 91
 - window 97
- education
 - information 84
 - World Wide Web 85
- entry field 94
- environment variables
 - COBOL GUI Designer 14
 - Compiler and Runtime 13
 - Data Assistant 14
 - Data Description and Conversion 15
 - Editor 14
 - Interactive Debugger 14
 - Performance Analyzer 14
 - Remote Edit/Compile 15
 - SMARTdata UTILITIES 15
 - Transaction Assistant 15
 - VisualAge COBOL information 14
 - Warp Toolkit 16
 - WorkFrame 15
- EPFIE114 27
- epfinsts 27
- error log, creating 31
- error message help 25
- error messages, build 109
- event-driven programming 57
- exiting the install program 12

F

- FAT file system 5
- first failure support technology (FFST/2) 170
- fixes, access 83

G

- getting help for installation 7
- glossary 195
- GUI
 - adding parts 94
 - building 65

GUI (continued)

- create application with 89
 - creating 49
 - creating application project 93
 - creating logic 57
 - customizing 53
 - customizing parts 94
 - part names 95
 - running application 112
 - saving 55
 - saving project 95
- GUI Code Assistant
 - copy file 62
 - description 69
 - using 61
 - GUI Designer, COBOL
 - coding event logic 59
 - description 49
 - saving a project 55
 - using 52

H

- hardware requirements 3
- headers and libraries, Warp
 - disk space 4
- help
 - installation 7
 - installation problems 25
- history log, creating 31

I

- IDBUG tool 110
- Information
 - disk space
 - Compiler and Runtime 4
 - environment variables 14
 - Information Notebook 80
- inheritance 69
- installing
 - a second time 19
 - additional components 22
 - application on end user machine 115
 - components 9
 - exiting the program 12
 - problems and error messages 25
 - second phase 12, 17
 - specifying target drive and directory 10
 - stopping the procedure 10

- installing (*continued*)
 - to workstation 7
 - unattended install 16
 - using a LAN
 - description 5
 - preparing the server 5
 - with a software distribution manager 18
- Interactive Debugger
 - description 71
 - disk space 4
 - environment variables 14
 - using 110
- IWZINST.LOG file 17
- IWZVFAZ2 command 26

K

- keywords for response files 29

L

- LAN installation
 - administrator tasks 5
 - description 5
 - preparing the server 5
- LAN to NCP definition 182
- LAN to VTAM definition 180
- language sensitive editing, specifying 91
- license agreement 5
- lockup 110
- logic
 - calling subroutine 100
 - creating application 97
 - creating subroutine 91
- LU definition 181

M

- mainframe concepts, compared with workstation 199
- memory requirements 3
- message boxes, CM/2 - checking values 169
- monitor window 108
- monitoring APPC 183
- multimedia bitmaps, Warp
 - disk space 4
- MVS, configuring communications 150

N

- NCP
 - defining 3745 Attached LAN 182
 - defining token-ring adapter 182
- nesting projects 103
- new project, creating 90

O

- object file for subroutine 91
- online help, using 80
- operating system requirements 3
- options, command-line 31
- ordering documentation 81
- OS/2
 - using 131
- overview
 - CICS configuration 186
- overwriting files 8, 22

P

- packaging application 113
- part names, GUI 95
- Performance Analyzer
 - description 72
 - disk space 4
 - environment variables 14
- phase 2 install 12, 17, 26
- PostScript files 81
- preinstallation tasks 3
- prerequisites
 - APPC/MVS 170
 - CICS 184
 - CM/2 150
- printing documentation 81
- problems during installation 25
- profiles in Communications Manager 150
- programmable workstation
 - host communications, configuring 149
- project
 - building a GUI target 65
 - building a target 47
 - correcting syntax errors 47
 - creating 43
 - creating GUI 93
 - creating new 90
 - definition 40
 - nesting 103

- project (*continued*)
 - running a target 48
 - saving, GUI 95
 - subroutine 90
 - WorkFrame 69
- PU definition 181
- push button field 94

Q

- quick reference 75
- quitting the install program 12

R

- RAM, minimum 3
- README file 7
- reinstalling components 19
- remote Debug Tool 167
- Remote Edit/Compile
 - configuring 149
 - disk space 4
 - environment variables 15
 - REXX procedures 193
- requirements
 - hardware 3
 - NLS 3
 - software 3
- response file
 - adding and deleting components 23
 - CID support 18
 - creating 17
 - nesting 30
 - required keywords 30
 - specifying on command line 32
 - syntax 29
- return codes 32
- returning to Communications Manager profile list 164
- REXX procedures
 - compile and link 193
- running
 - application, GUI 112
 - install a second time 19

S

- sample application
 - building 43
 - Data Conversion Utility samples 148
 - Employee Lookup Application samples 139

- sample application (*continued*)
 - running 48
 - SMARTdata Utilities 147
 - SMARTsort samples 148
 - source code 45
 - workstation VSAM samples 147
- sample GUI application
 - building 49
 - running 67
- sample programs, Warp
 - disk space 4
- saving
 - GUI project 95
- second-phase install 12, 17, 26
- selecting components to install 9
- server-requester communications, configuring 149
- server, setting up as installation base 5
- service
 - CompuServe 83
 - FAX number 84
 - fixes 83
 - information 83
 - mail address 84
 - voice support 83
 - World Wide Web 84
- setting
 - compiler options 103
 - target directory 10
- settings notebook
 - changing attributes 95
- SMARTdata UTILITIES
 - description 71
 - disk space 4
 - environment variables 15
 - prompt 11
 - sample applications 147
 - SMARTsort Defaults window 11
- SNA features list 164
- SNA LU 6.2 150
- software distribution manager (SDM) install
 - return codes 32
- software requirements 3
- source code
 - example of application 101
 - example of subroutine 92
 - saving 92
- starting new project 90
- static text field 94
- stopping the installation 10

- subroutine
 - calling 100
 - creating 90
 - object file 91
- swap file growth 5
- swapper, disk space 4
- synchronous communications 150
- syntax errors, correcting 47

T

- target
 - project 91
- target drive and directory
 - determining
 - available space 10
 - setting 10
- task reference 75
- terminology
 - APPC and APPC/MVS 171
 - APPC and Communications Manager 150
 - CICS 184
- training, VisualAge COBOL 84
- Transaction Assistant
 - description 70
 - disk space 4
 - environment variables 15
 - tutorial 127
- Transaction Programs (TP), defining 177
- tutorial
 - Data Assistant 117
 - Tax Calculation subroutine 90
 - Tax Computation Application 89
 - Transaction Assistant 127

U

- unattended install
 - procedure 16
 - UNATTEND.RSP file 17
 - with a software distribution manager 18
- uninstalling components 20
- updating CONFIG.SYS 8

V

- VACCESS.CPY copy file 62
- Visual tools
 - COBOL GUI Designer 4
 - Data Assistant 4

- Visual tools (*continued*)
 - Data Description and Conversion 4
 - Transaction Assistant 4
 - WorkFrame 4
- VisualAge COBOL
 - building a GUI application 49
 - building an application 43
 - COBOL GUI Designer 70
 - compiler 70
 - concepts 40
 - consulting services 84
 - creating a GUI 49
 - creating a new project 43
 - Data Assistant 70
 - description 37
 - documentation 79
 - Editor 69
 - education 84
 - GUI Code Assistant 69
 - Information Notebook 80
 - Interactive Debugger 71
 - learning path 79
 - online help 80
 - ordering documentation 81
 - Performance Analyzer 72
 - printing documentation 81
 - quick reference 75
 - run-time 70
 - service 83
 - Transaction Assistant 70
 - tutorials 87
 - WorkFrame 69
- VM, configuring communications 150
- VSAM data 139, 146
- VTAM
 - defining 3745 Attached LAN 180
 - defining APPC/MVS LU 175

W

- Warp Toolkit
 - Development tools, disk space 4
 - environment variables 16
 - headers and libraries 4
 - multimedia bitmaps 4
 - sample programs 4
- WorkFrame 69
 - disk space 4
 - environment variables 15

workstation concepts, compared with mainframe 199

World Wide Web

consulting services 84

education 85

fixes 83

service 84

X

XCOPY command 5

We'd Like to Hear from You

IBM VisualAge for COBOL for OS/2

Getting Started

Publication No. GC26-8421-01

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 STLVM27
 - Internet: COMMENTS@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

IBM VisualAge for COBOL for OS/2

Getting Started

Publication No. GC26-8421-01

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"/>

Please tell us how we can improve this book:

May we contact you to discuss your comments? Yes No

Name

Address

Company or Organization

Phone No.

Readers' Comments
GC26-8421-01

IBM®

Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



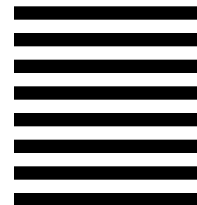
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945



Fold and Tape

Please do not staple

Fold and Tape

GC26-8421-01

Cut or Fold
Along Line

IBM®

Part Number: 33H3035

Printed in U.S.A.

