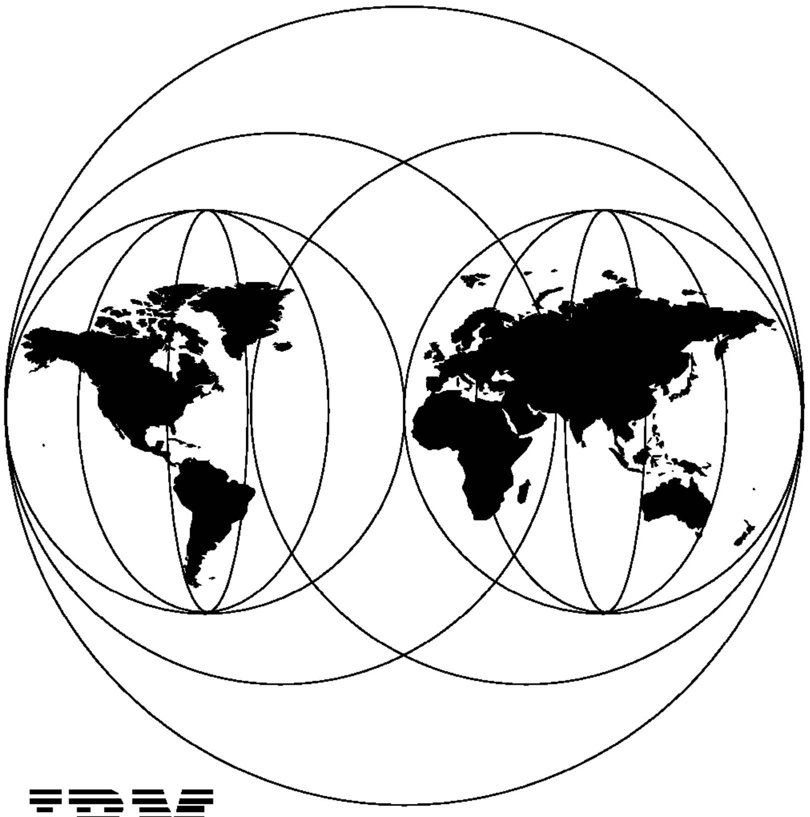
SG24-2245-00

VisualAge 2000 - Remote Edit, Compile, and Debug Using VisualAge COBOL 2.0 on OS/2

September 1997



IBM

International Technical Support Organization
San Jose Center

International Technical Support Organization SG24-2245-00

VisualAge 2000 - Remote Edit, Compile, and Debug Using VisualAge COBOL 2.0 on OS/2

September 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 103.

First Edition (September 1997)

This edition applies to Version 2.0 of VisualAge for COBOL, Standard Edition for OS/2, 4226000, for use with the operating systems:

- OS/2 WARP 3.0 (with FixPak 26)
- OS/2 WARP 4.0 (with FixPak 1)

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

Contents

Figures . . . v

Preface . . . vii
The Team That Wrote This Redbook . . . vii
Comments Welcome . . . viii

Chapter 1. Introduction . . . 1

Chapter 2. Remote E/C/D with TCP/IP and NFS on OS/2 . . . 3
2.1 Configuring Communications for TCP/IP . . . 4
2.1.1 Configuration of TCP/IP on the Workstation . . . 5
2.1.2 MVSINFO.DAT File Explanation . . . 12
2.2 Accessing the Host from the Workstation . . . 17
2.3 Configuring the Host for TCP/IP and File Mapping . . . 21
2.4 Upload Workstation Files Using FTP . . . 25

Chapter 3. Remote Edit, Compile, and Debug with APPC on OS/2 . . . 29
3.1 Configuring Communications for APPC . . . 30
3.1.1 Installing Communications Manager/2 for OS/2 Warp . . . 31
3.1.2 Configuring Communications Manager/2 for OS/2 Warp . . . 40
3.2 Set up the PC to Connect to the Host . . . 51
3.2.1 Configuration of SMARTdata Utilities on the Workstation . . . 53
3.3 Configuring the Host for APPC . . . 54
3.4 Accessing the Host from the Workstation . . . 55

Chapter 4. Creating Applications Using Remote E/C/D on OS/2 . . . 59
4.1 Running a Sample Application on the Workstation . . . 59
4.1.1 Defining the Application on the Workstation . . . 59
4.1.2 Editing, Compiling, and Running the Application Remotely . . . 69
4.1.3 Debugging the Application . . . 76
4.2 Summary of Debugger Functionality and Additional Features . . . 82
4.3 DB2 Sample . . . 84
4.3.1 Create a New Database on the Host . . . 85
4.3.2 Create JCL Members to Compile the COBOL Program . . . 89
4.3.3 Creating the Project and Running the DB2 Sample . . . 90

Chapter 5. MVS Data Types Sample . . . 93

Appendix A. DFM/MVS DataAgent . . . 99

Appendix B. Contents of the Enclosed Diskette . . . 101

Appendix C. Special Notices . . . 103

Appendix D. Related Publications . . . 105
D.1 International Technical Support Organization Publications . . . 105
D.2 Redbooks on CD-ROMs . . . 105
D.3 Other Publications . . . 105

How to Get ITSO Redbooks . . . 107
How IBM Employees Can Get ITSO Redbooks . . . 107

How Customers Can Get ITSO Redbooks 108
IBM Redbook Order Form 109

Glossary 111

List of Abbreviations 131

Index 133

ITSO Redbook Evaluation 135

Figures

1. Configure Network Interface Parameters Window . . . 6
2. Configure Routing Information Window . . . 7
3. Route Entry Window – Default . . . 8
4. Route Entry Window – Net . . . 8
5. Configure LAN Name Resolution Services Page . . . 9
6. Nameserver Entry Window . . . 9
7. Configure Name Resolution Services Page . . . 10
8. HOSTS Entry Window . . . 10
9. Editor – M:\MVS File System Access Monitor Window . . . 19
10. Drives – Icon View Window . . . 19
11. NFS.MAPPING File . . . 24
12. FTP-PM – Open Remote Host Window . . . 25
13. FTP-PM - wtsc47 (cobrs08) Window . . . 26
14. FTP-PM - Put Local Files Window . . . 27
15. FTP-PM Window . . . 27
16. Installation Window for Communications Manager 2 . . . 31
17. Target Drive Selection Window . . . 32
18. Communications Manager Setup Window . . . 32
19. Copying Files Window . . . 33
20. Open Configuration Window . . . 33
21. OS/2 Communications Manager Window—Workstation . . . 34
22. Communications Manager Configuration Definition Window . . . 34
23. 3270 Emulation through Token-ring Window . . . 35
24. OS/2 Communications Manager Window—Product Files . . . 35
25. Install Window—Communications Manager/2 for OS/2 Warp . . . 36
26. Change CONFIG.SYS Window . . . 36
27. Communications Manager Completion Window . . . 37
28. Communications Manager/2 - Icon View Window . . . 37
29. Multi-Protocol Transport Services Window . . . 38
30. Configure Window—MPTS . . . 38
31. LAPS Configuration Window . . . 39
32. Update CONFIG.SYS Window . . . 39
33. Open Configuration Window for APPC . . . 41
34. OS/2 Communications Manager Window for APPC . . . 41
35. Communications Manager Configuration Definition Window . . . 42
36. Communications Manager Profile List Window for APPC . . . 42
37. Token Ring or Other LAN Types DLC Adapter Parameters Window . . . 43
38. Local Node Characteristics Window . . . 44
39. Connections List Window . . . 44
40. Adapter List Window . . . 45
41. Connection to a Host Window . . . 46
42. Partner LUs Window . . . 47
43. OS/2 Communications Manager Window (3270 emulator information) . 47
44. SNA Features List Window . . . 48
45. Local LU Window . . . 48
46. Mode Definition Window . . . 49
47. Transaction Program Definition Window . . . 50
48. Additional TP Parameters Window . . . 50
49. Communications Manager—Checking Values Window . . . 51
50. Create New Project - Catalog View Window . . . 60
51. Project Smarts - Console Window . . . 61

52. Project Smarts - Variable Settings Window 61
53. Project Smarts - Target Information Window 62
54. Done! Window 62
55. MVS Project Hello - Settings Window 63
56. MVS Project Hello - Settings Window: Location Page 64
57. Create directories Window 65
58. MVS Project Hello - Settings Window: Inheritance Page 65
59. MVS Project Hello - Icon view Window 68
60. MVS Project Hello - Icon View Window with Pop-up Menu 69
61. Editor - M:\HELLO\COBOL\hello.cbl Window 70
62. Change profile Window 71
63. MVS Job Status Monitor Window 73
64. Editor - M:\Held SYSOUT output for job Window 74
65. Waiting to connect Window 78
66. Debugger - Session Control Window 78
67. Source Filename Window 78
68. Source: DTTEST – Thread:1 Window 79
69. Source: DTTEST - Thread:1 Window – Start of DTTEST 79
70. Program Monitor Window for DTTEST 80
71. Source: HELLO - Thread:1 Window 81
72. Program Monitor Window for HELLO 81
73. Application Exception Action Window 82
74. Debugger Message Window 82
75. Call Stack Window 83
76. Registers Window 83
77. Storage Window 83
78. Breakpoints List Window 84
79. The Part of the JCL to Be Modified 87
80. Bind and Execute Statements of DSNTIAD 87
81. MVS Data Type - Icon view Window 93
82. MVS Data Types - Tools setup Window 94
83. Change Environment Variable Window 95

Preface

This redbook details the configuration of the communication between workstation and host systems to use the new remote edit, compile, and debug component of VisualAge for COBOL Version 2 on OS/2. Installation details are covered as well as application development techniques. Several practical examples are provided on diskette for you to install on your host and use with the book. The redbook gives you an introduction to developing host applications from a workstation environment and explains in detail how to configure your workstation for communicating with the host via TCP/IP. You are led through the setup of the TCP/IP as well as of Network File System (NFS), the file system you need.

The redbook describes how to configure your workstation for communicating with the host via APPC, the configuration of the Communications Manager/2 for using APPC and SMARTdata Utilities, and the changes you have to make on the workstation side as well as on the host side are explained.

This redbook also covers the configuration of a WorkFrame project and the use of the project files by use of a sample program. In this sample, you learn step by step how to create a project and how to edit, compile, and debug the programs. A more complex project with DB2 access is shown also. You are also shown, by means of a sample, how to use VSAM for the workstation for local and remote data access, including customizing data description and conversion for transparent remote data access.

Knowledge of application development in an MVS environment is a prerequisite for the MVS portions of this book. Knowledge of COBOL programming and OS/2 is presumed. Some knowledge of TCP/IP and APPC would also be helpful.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from Europe working at the International Technical Support Organization San Jose Center.

Ute Lotz, IBM Germany (Residency Team Leader)

Ernesto Carretta, 3I, Italy (IBM Business Partner)

Thanks to the following people for their invaluable contributions to this project:

Neal Eisenberg
IBM Santa Teresa Laboratory, San Jose

Mickey Forman
IBM Santa Teresa Laboratory, San Jose

Bob Haimowitz
IBM International Technical Support Organization, Poughkeepsie Center

Milton Hall
IBM Santa Teresa Laboratory, San Jose

Wilbert Kho
IBM Santa Teresa Laboratory, San Jose

Barbara Price
IBM Santa Teresa Laboratory

Koko Yamaguchi
IBM Santa Teresa Laboratory

This project was designed and managed by:

Joe DeCarlo
IBM International Technical Support Organization, San Jose Center

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 135 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

For Internet users	`http://www.redbooks.ibm.com`
For IBM Intranet users	`http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. Introduction

VisualAge for COBOL provides a new way of developing host applications with the common business oriented language (COBOL). The remote edit. compile, and debug feature lets you maintain your host applications from your workstation.

This function provides a seamless workstation development environment for the development of host programs. Files on the host are directly accessed from the workstation without the need to replicate the host environment on the workstation and, therefore, without the need for a workstation naming convention and mapping.

Remote edit/compile/debug is integrated with workstation components such as the live parsing editor (LPEX) and WorkFrame. That ensures that the host has a working environment similar to the one you have on your personal computer (PC). Thus, you can work with these components, without needing to learn the handling of a new editor or other tools.

The remote VisualAge COBOL debugger helps you to debug applications residing on the host directly from your workstation. Remote debugging features the use of a graphical user interface (GUI) when interacting with the host debug tool.

The advantage of this tool is that the user interface is similar to that of the debugger on the PC, offering the same handling, the choice to set breakpoints, the opportunity to follow the change of specific variables, and the chance to set up new values for these variables in the way that the program continues using the modified variables. In this way, you can affect the further execution of the program and you can change the variables to run special statements that would be inaccessible otherwise.

For the communications to the host you can use the transmission control protocol/internet protocol (TCP/IP) or the advanced program-to-program communication (APPC) protocol.

This new functionality of VisualAge for COBOL for OS/2 (operating system/2) and for Windows NT (Microsoft Windows NT; where NT stands for ″new technology″) offers you two ways of developing and redeveloping host programs. For the Windows NT environment obtain Redbook SG24-2250, *VisualAge 2000 - Remote Edit, Compile, and Debug Using VisualAge for COBOL V2.0 on Windows NT*. Either you offload your existing COBOL programs from the host to modify them on the workstation, or you use the remote edit, compile, and debug component to work remotely on the host. Each of these working methods has its own advantages so you should decide which way you want to go. It is not a general decision that has to be made. It can vary from project to project because it depends on several aspects.

Here are some considerations you have to take into account:

- An important factor is the response time of the host. If the host does not respond properly, it is better to use the power of the PC for editing and debugging the programs.
- If you make only small changes on the host programs and only a short amount of time is planned for this project, use the remote edit, compile, and

debug facility instead of down-loading the files to the PC first and setting up an environment that mimics the host environment.

- Another important aspect is how many programs or files are already residing on the host and would have to be down-loaded and how many programs or modules have to be created on the workstation.
- You may want to use the same tools that you have on the PC, but you do not want to develop applications on the PC. Use the remote edit, compile, and debug functionality that VisualAge for COBOL offers you.

As you proceed through the chapters that follow, we are sure you will see the advantages of using the remote edit/compile debug feature of VisualAge for COBOL for host application development and maintenance from the workstation.

The remote edit, compile, and debug component is also referred to as *remote edit/compile/debug* and *remote e/c/d* throughout this book.

Chapter 2. Remote E/C/D with TCP/IP and NFS on OS/2

You have two different protocols to choose from in order to communicate from your workstation with the host. For each protocol, you must set up a different configuration on the host and on your workstation.

Section 2.1, "Configuring Communications for TCP/IP" on page 4 explains the configuration on the workstation and on the host for the use of the TCP/IP. Use of APPC is covered in Chapter 3, "Remote Edit, Compile, and Debug with APPC on OS/2" on page 29.

Before you start with any configuration, make sure that your host and workstation meet the software requirements for the use of the remote edit, compile, and debug component. What follows is the prerequisite software for the host and workstation environments for OS/2 as found in the product announcement. For the full product announcement,see announcement 297-142 on May 6, 1997, titled, *VisualAge for COBOL Version 2.0 Delivers Additional Support for Remote Host Development, Year 2000, and Windows Development*

```
To use VisualAge for COBOL Version 2.0 (Standard Edition)
on the OS/2 platform, one of the following operating systems
must be installed:

-   IBM OS/2 Warp Version 3.0 plus FixPak 26
-   IBM OS/2 Warp Version 4.0 plus FixPak 1

When installing VisualAge for COBOL Version 2.0 (Standard Edition)
be sure to install (in addition to the default components):

-   Remote Edit/Compile/Debug

To use host data access (via APPC) on OS/2, one of the following:

-   IBM Communications Manager/2 Version 1.11

To access VSAM/SAM files on your host (MVS or OS/390) with the
SMARTdata UTILITIES (SdU):

-   DFSMS/MVS (R) Version 1.2.0 or later is required on your host

To use remote edit/compile/debug between your host (OS/390 or
MVS) and OS/2 via APPC:

-   On the OS/390 host:

    --  IBM COBOL for OS/390 & VM Version 2 Release 1 Full
        Function Offering plus Debug Tool PTFs and the PTF for
        APAR PQ03533
    --  OS/390 Release 3 Language Environment feature
    --  DFSMS/MVS Version 1.3.0 or later and the PTF for APAR
        OW20884

-   On the MVS host:

    --  IBM COBOL for MVS & VM Release 2 Full Function Offering
        plus Debug Tool PTFs and the PTF for APAR PQ03512
```

    -- IBM Language Environment for MVS & VM Release 5 plus PTFs
    -- DFSMS/MVS Version 1.3.0 or later and the PTF for APAR OW20884

- On the OS/2 workstation:

    -- IBM Communications Manager/2 Version 1.11 or later

To use remote edit/compile/debug between your host (OS/390 or MVS) and OS/2 via TCP/IP:

- On the OS/390 host:

    -- TCP/IP Version 3 Release 2 for MVS/ESA (TM)
    -- IBM COBOL for OS/390 & VM Version 2 Release 1 Full Function Offering plus Debug Tool PTFs and the PTF for APAR PQ03533
    -- OS/390 Release 3 Language Environment feature
    -- DFSMS/MVS Version 1.2.0 with the Network File Feature (minimum), or, DFSMS/MVS Version 1.3.0 or later with the Network File System Feature and the PTF for APAR OW25973 (recommended)

- On the MVS host:

    -- TCP/IP Version 3 Release 2 for MVS/ESA
    -- IBM COBOL for MVS & VM Release 2 Full Function Offering plus Debug Tool PTFs and the PTF for APAR PQ03512
    -- IBM Language Environment for MVS & VM Release 5 plus PTFs
    -- DFSMS/MVS Version 1.2.0 with the Network File Feature (minimum), or, DFSMS/MVS Version 1.3.0 or later with the Network File System Feature and the PTF for APAR OW25973 (recommended)

- On the OS/2 workstation:

    -- IBM TCP/IP for OS/2 Version 2.0 or later
    -- IBM TCP/IP for OS/2 Version 2.0 NFS Kit with CSD UN57064 and the fix for APAR PQ00835

Once you have all the prerequisite software installed on your host and workstation you can proceed with the communication configuration.

2.1 Configuring Communications for TCP/IP

This section explains how to configure the TCP/IP of your workstation to be able to connect to a host server directly without the necessity of taking care of any other intervening servers.

In the following, we help you set up your workstation for communication with the host via TCP/IP.

TCP/IP for OS/2 provides extensive facilities for communicating over an internet. With TCP/IP for OS/2 installed on your workstation, you can perform the following communication tasks:

- Log on to a remote host
- Transfer files between hosts
- Print files using a central printer server

- Send and receive electronic mail
- Run commands on a remote host.

TCP/IP for OS/2 is packaged in a Base Kit and several component kits. The Base Kit provides the protocol stack necessary to support all the functions of TCP/IP for OS/2. It also provides a base set of applications, including Telnet, file transfer program (FTP), and Sendmail.

For your requirements, it is sufficient to have only the TCP/IP Base Kit installed.

Network file system (NFS) is the file system designed for communications to access the host files. It allows users to access files and run programs located on remote systems as if they were local.

An NFS client uses the MOUNT protocol to request that a directory be mounted on the server machine, or made available to the client. Once a directory is mounted, the client can list, read, write, and run remote files or programs in the directory as if they existed locally.

NFS uses remote procedure call (RPC) and the user datagram protocol (UDP) for communication between clients and servers.

The NFS server is designed so that the client will continue to operate, even though the server might go down and be forced to restart. The client will continually resend a request for data until the server is able to respond. This type of operation is called *stateless*, since neither the client nor the server needs to check connection status during operation.

To use NFS, you must have the TCP/IP stack from either the TCP/IP for OS/2 Base Kit or from some other product installed on your workstation. The next step is to configure your TCP/IP as described in the next subsection.

2.1.1 Configuration of TCP/IP on the Workstation

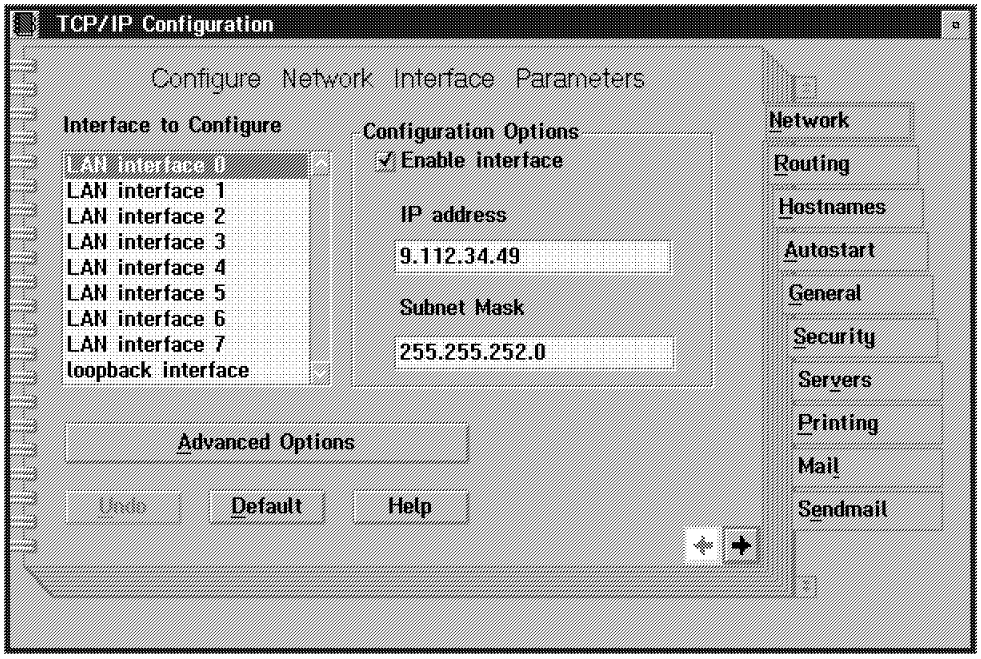
At first, you have to configure the TCP/IP of OS/2 Warp. It comes with OS/2 Warp Connect and need not be installed separately, when you already have installed OS/2 with the default configuration. To check whether the TCP/IP is installed on your PC or not, open the OS/2 System folder by double clicking on its icon on the desktop and check whether if the TCP/IP icon is in it. If not, install it from the OS/2 CDROM and reboot your PC.

Also, you have to install the product that is called NFS for TCP/IP on OS/2 2.0 and 2.1. Use version 2.0 with the corrective service diskette (CSD) level UN57064 or higher. To check if the right version and level is installed, type `SYSLEVEL` in an OS/2 window and you get the levels of all products you have installed on your workstation running on OS/2. When the syslevel information for NFS appears, check the number of the current CSD level.

Now the workstation is ready to be configured for TCP/IP:

1. Open the OS/2 System folder and open the TCP/IP folder by double clicking on the appropriate icon. Then double-click on the **TCP/IP Configuration** icon in the **TCP/IP folder**. The TCP/IP Configuration notebook appears (Figure 1 on page 6) in which you should do all your configurations for the TCP/IP of OS/2.

   On this first page (with the *Network* tab) that is called *Configure Network Interface Parameters* you can modify the network configuration of your primary adapter, or create network configurations for up to seven additional adapters. Here you have to specify the internet protocol (IP) address of your workstation and the subnet mask.



✓ Enable interface

IP address: 9.112.34.49

Subnet Mask: 255.255.252.0

Figure 1. Configure Network Interface Parameters Window

If you do not know your IP address, ask your system administrator for it. The IP address of each workstation should be unique in the world but it must at least be unique in the local network of your company.

2. Scroll to the next tab to configure the routing information. If the local area network (LAN) to which your workstation is attached is connected to other LANs through routers or gateways, you must configure your TCP/IP to recognize these routers or gateways.

   If you already have a connection configured, this page is set up and you do not need to change anything. Otherwise, you need to know the IP address of the router to specify the routing table information here, as shown in Figure 2 on page 7.

   If, however, you are not really sure that the given router is yours, try to verify its availability. For that, open an OS/2 window and type

   ```
   ping routernumber
   ```

   where router number is the IP address of the router specified in Figure 2 on page 7.

   This command directs a tool called *packet internet groper*(PING) to send echo requests to the foreign router. When the ping is successful, you get a report like the following:

   ```
   PING 9.112.32.5: 56 data bytes
   64 bytes from 9.112.32.5: icmp_seq=0. time=8. ms
   64 bytes from 9.112.32.5: icmp_seq=1. time=0. ms
   64 bytes from 9.112.32.5: icmp_seq=2. time=0. ms
   ```

PING displays a line of information for each echo request. These messages come up continuously until you cancel the request by pressing Ctrl-C. Then you get a statistic about the connection like this:

```
----9.112.32.5 PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/2/8
```

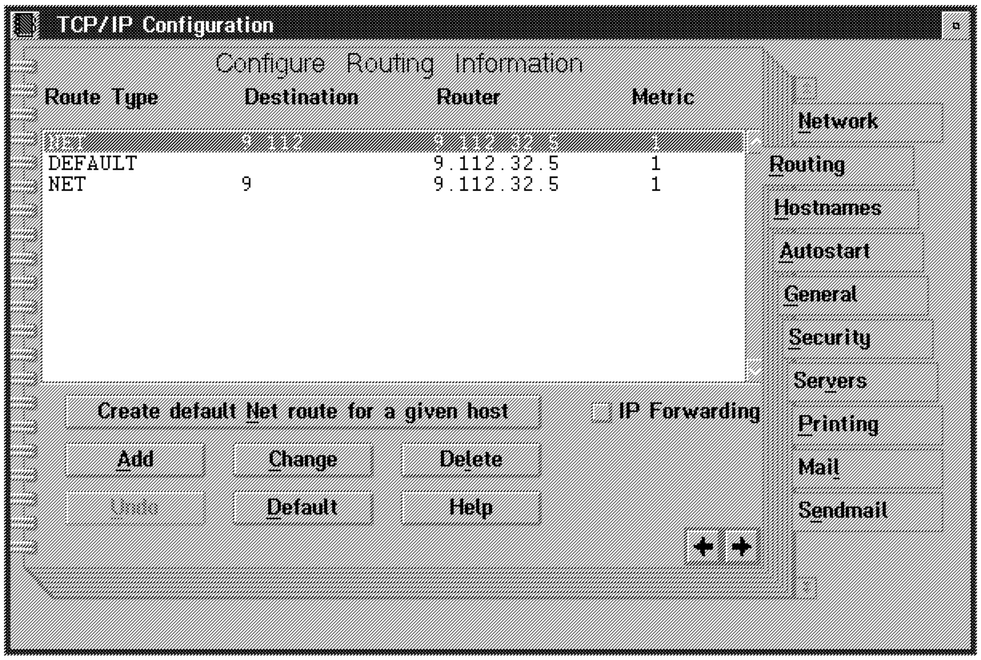
and you know that the connection works.

If the connection to the router fails, you get only the first ping message. You get no further message about any transferred packages and no statistic afterward.

Ping

PING is a diagnostic tool that sends an echo request to a foreign host to determine whether the computer is accessible. The echo request sent by the ping command does not guarantee delivery. More than one ping command should be sent before you assume that a communication failure has occurred.

We recommend that you try this ping just to ensure that the defined router is the right one.

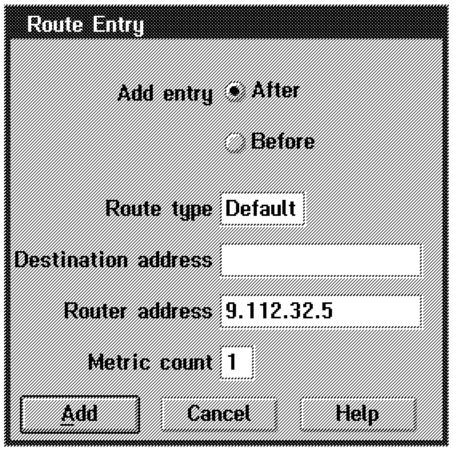


Route Type	Destination	Router	Metric
NET	9.112	9.112.32.5	1
DEFAULT		9.112.32.5	1
NET	9	9.112.32.5	1

Figure 2. Configure Routing Information Window

3. If the routing information is not shown on the Configure Routing Information page, click on **Add** to insert the necessary router. The **Route Entry** window comes up.

   Delete the entry from the field *Route type* and type a `D` in it (for default). The system completes it immediately to Default. Leave the *Destination address* field empty and type the number of the router in the *Router address* field. In our environment the router number is 9.112.32.5. Ensure that the *Metric count* is 1, as shown in Figure 3 on page 8.



Add entry ● After ○ Before

Route type Default

Destination address

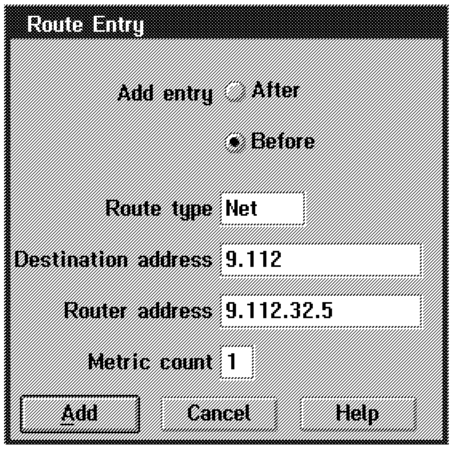
Router address 9.112.32.5

Metric count 1

Figure 3. Route Entry Window – Default

4. The Configure Routing Information page (Figure 2 on page 7) got two entries. You see the route type default and the route type net. Now select the *Route Type* **Default** and click on **Add** again. The **Route Entry** window appears again. Enable **Before** for the *Add entry* check-box. Ensure that the *Route type* is **Net** and give the *Destination address*. Our destination address is 9.112. Then type the same router address as before into the *Router address* field. The *Metric count* is 1 again (Figure 4).

   Click on **Add** to complete the routing information. Your Configure Routing Information page should have three entries, as shown in Figure 2 on page 7.



Add entry ○ After ● Before

Route type Net

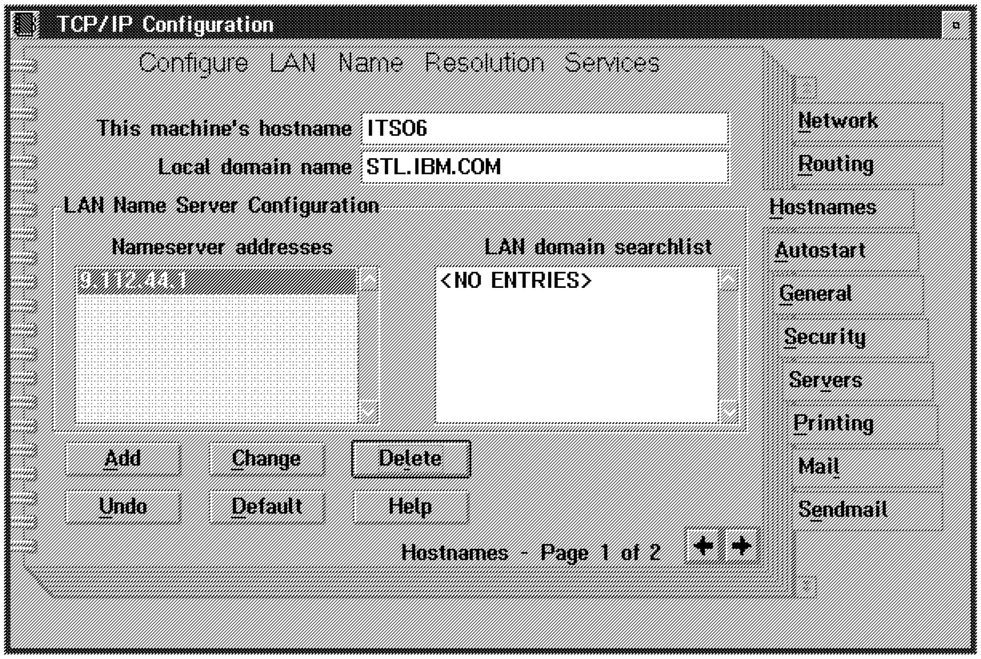
Destination address 9.112

Router address 9.112.32.5

Metric count 1

Figure 4. Route Entry Window – Net

5. Now scroll to the *Hostnames* tab to see the page Configure LAN Name Resolution Services (Figure 5 on page 9).


TCP/IP Configuration

Configure LAN Name Resolution Services

This machine's hostname: ITSO6

Local domain name: STL.IBM.COM

LAN Name Server Configuration

Nameserver addresses: 9.112.44.1

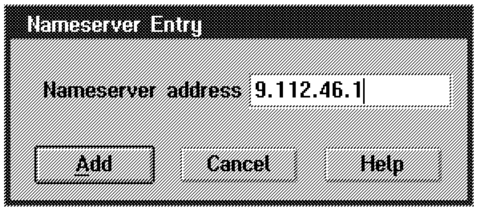
LAN domain searchlist: <NO ENTRIES>

Figure 5. Configure LAN Name Resolution Services Page

Indicate the name of the machine's host name in the field *This machine's hostname*. This can be seen as an alias for the IP address and can be used instead of it when connecting. In the field called *Local domain name*, you should specify the domain where the host resides.

To connect to a domain, the domain has to be known by a name server. The name servers resolve domain names to IP addresses. For that, you can specify domain name servers in the *Nameserver addresses* field. To do so, click on the *Nameserver addresses* field to activate the push-buttons below and then select the **Add** push-button to add one name server address.

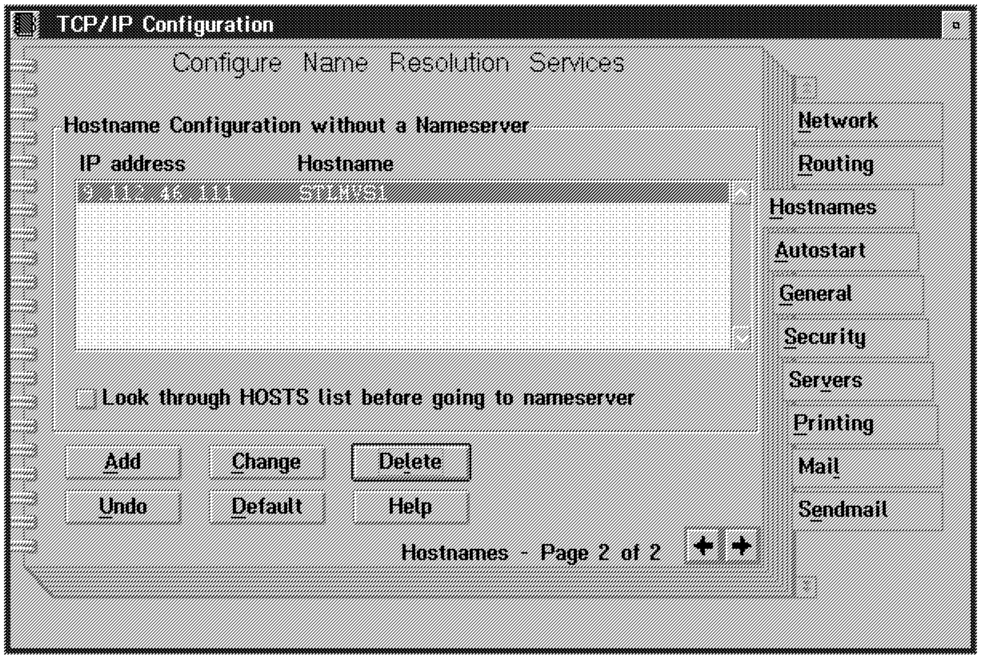
6. The **Nameserver Entry** window comes up. Type the IP address you want to add to the list into the *Nameserver address* field and click on **Add** to leave this window (Figure 6).


Nameserver Entry

Nameserver address: 9.112.46.1

Figure 6. Nameserver Entry Window

7. Go to page two of the *Hostnames* tab to specify the IP addresses and their corresponding host names (Figure 7 on page 10).



Hostname Configuration without a Nameserver

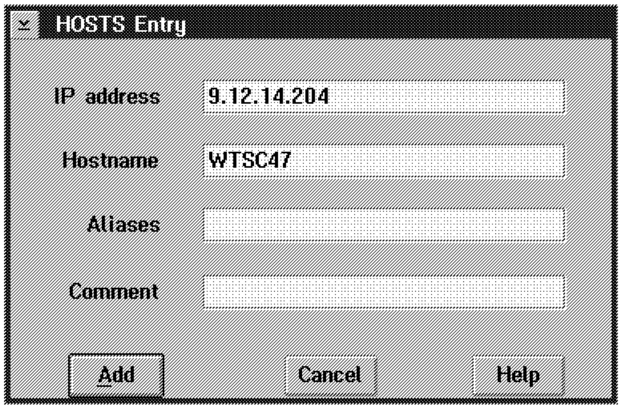
IP address	Hostname
9.112.46.111	STLMVS1

☐ Look through HOSTS list before going to nameserver

Figure 7. Configure Name Resolution Services Page

This resolution table indicates which host name belongs to which IP address. When you specify it here, you can work without being connected to the name server at any time and you can leave out the name server information on the previous page. But we recommend using the name server for the resolution of the IP addresses if possible.

To add a host name, select the *Hostname Configuration without a Nameserver* list and click on **Add**. The **HOSTS Entry** window comes up, in which you enter the IP address and its corresponding host name into the appropriate fields (Figure 8).



IP address	9.12.14.204
Hostname	WTSC47
Aliases	
Comment	

Figure 8. HOSTS Entry Window

Giving an alias or a comment is optional. Click on **Add** to insert the new host name into the configuration list on the page Configuration Name Resolution Services. Finally, ensure that the check-box *Look through HOSTS list before going to nameserver* is disabled.

Note

If you intend to step through the following pages in the **TCP/IP Configuration** window and you reach the Configure NFS Parameters page, do not try to add the mount command you will use later. If you add it here, it causes problems with the connection later.

8. Close the **TCP/IP Configuration** window by double clicking on the icon in the upper left corner of the window. The window **Closing TCP/IP Configuration** comes up and asks you if you want to save your changes. Click on **Save** to save the changes you have made.

   The TCP/IP configuration is now set up.

9. To test if the TCP/IP connection to the host is working, open an OS/2 window and write

   ```
   ping hostname
   ```

   where hostname is the name that belongs to the IP address of the host you want to connect. You specified this hostname in Figure 7 on page 10.

   If the host is responding, you get messages like the following:

   ```
   PING wtsc47: 56 data bytes
   64 bytes from 9.12.14.204: icmp_seq=0. time=125. ms
   64 bytes from 9.12.14.204: icmp_seq=1. time=109. ms
   64 bytes from 9.12.14.204: icmp_seq=2. time=109. ms
   64 bytes from 9.12.14.204: icmp_seq=3. time=109. ms
   64 bytes from 9.12.14.204: icmp_seq=4. time=109. ms
   64 bytes from 9.12.14.204: icmp_seq=5. time=109. ms
   ```

   This response tells you that the connection is running properly. You can cancel the connection now by pressing Ctrl-C. The connection stops and you get statistics about the connection you had with the host. Important to these statistics is the percent of the transmitted packets that got lost (zero here):

   ```
   ----wtsc47 PING Statistics----
   6 packets transmitted, 6 packets received, 0% packet loss
   round-trip (ms)  min/avg/max = 109/111/125
   ```

10. Now you have to install the latest driver of NFS.

    Ensure that you have Version 2.0 of NFS for TCP/IP with the CSD level UN57064 and install the very latest fix for authorized program analysis report (APAR) PQ00835 (available on the Internet as this book is being published.)

    Go to the Web site **http://service2.boulder.ibm.com/psppaper**, select **All Closed APARs**, and then search on PQ00835 to get this APAR. From this site you can download the latest fixes that are not yet on a CSD.

    To install APAR PQ00835, follow the instructions contained in the READ.ME file that comes with APAR PQ00835.

    If you do not have the CSD level UN57064, go to the Internet and load the uniform resource locator (URL) **http://www.networking.ibm.com**, select **Software Support, Technology, Products**, select **TCP/IP Leadership**, select **OS/2 Warp**, select **Fixes**, and then select **Fixes for TCP/IP for OS/2** and look for the CSD UN57064.

    After the installation, shut down the workstation.

11. Before continuing, start NFS by typing

    ```
    NFSSTART
    ```

in an OS/2 window and connect to your host system from the command line to ensure that NFS is working successfully. To mount a drive, type

```
mount m: hostname:"yourname,text,crlf"
```

where hostname is the name you entered in the TCP/IP configuration for the IP address and yourname stands for your user ID on this host.

While making the host connection, the system prompts for both your user ID (called *user name*) and your password. If the command fails, data facility storage management/multiple virtual storage (DFSMS/MVS) on multiple virtual storage (MVS) or operating system/390 (OS/390) might not be installed and running on your host system or personal computer network file system daemon (PCNFSD) might not be started. If you receive a prompt for a group ID (GID) as well as your user ID, it is likely that PCNFSD is not running on the host.

Mount

When you issue the mount command, it issues a PCNFSD authentication request to establish the connection between your client and the server. When you issue the unmount command, the server flushes the data sets to DASD and breaks the connection between the mount point on the client and the server. The connection between the client and the server is automatically ended when the value specified in the logout attribute is reached.

If the connection to the host was successful, disconnect before continuing with the next section. To disconnect, type the command

```
umount m:
```

in the OS/2 window.

12. If you established the connection to the host established in the previous step, try connecting with file-extension mapping. To specify that you want file-extension mapping, change the mount command in the following way:

```
m: hostname:"yourname,text,crlf,fileextmap
```

If this command fails, file extension mapping is not available on the host.

To complete the configuration on the workstation to communicate with the host via TCP/IP you must set up a file that contains all the necessary information related to the project on the host that you want to access remotely. While starting the connection from the workstation to the host, the information is read, the connection to the specified host with the specified user ID is established, and the data sets given in the file are connected. After this successful connection, you can edit, compile, and debug all the COBOL sources that reside in the specified data sets on the host.

2.1.2 MVSINFO.DAT File Explanation

All the parameters needed to configure the connection from the workstation to a specified host can be set up in the MVSINFO.DAT file of VisualAge for COBOL. This file is located on the same drive on which you have installed VisualAge for COBOL for OS/2. It is in the directory /IBMCOBOL/MACROS. To edit the file, open an OS/2 window and type

```
IWZSET
```

Edit this file and set all the options that you need to configure the connection as described below.

The nonblank lines contained in the file are of two different types:

1. When the first character of the line is an asterisk, the line is a comment line and you should read it attentively when you configure it for the first time.
2. Otherwise, the line is empty or the first word of the line is a reserved word that specifies the characteristics of the command.

Here is the list of the reserved words and the action that must be done to configure the environment:

system	Change the name that follows this command with the name of the host to which you want to connect. This is the host name you specified in the TCP/IP configuration. The host we want to access has the host name WTSC47, so we specified system WTSC47.
worksys	No action required. This word is reserved for future use.
userid	Replace the name that follows with the user ID you use to connect to the host. We are using COBRS08 and changed the command to `userid COBRS08`.
pwd	Replace the name that follows with the encrypted password of the user ID you use for connection to the host. Further details on how to encrypt the password can be found in the listing. Instead of the encrypted password, you can specify that you want to be prompted to give the password manually. In this case, insert the string `pwd ++++++++ 1` instead of typing the encrypted password. We are using the password ++++++++, so we specified `pwd ++++++++` in this line.
filesys	Specify the system you are using to access the host files. We changed the options to `filesys nfs accessmon testaccess`.
readtimeout	No action required. We changed it to 10 to get better performance.
writetimeout	No action required. We changed it to 10 to get better performance.
nfs	No action required.
sdu	No action required. This word is reserved for future use.
drive	This statement supported by NFS allows you to access the data sets on the host in the same way as you would access a drive on your workstation. Specify here the high-level qualifier you want to use on the host and combine it with drive letters like the drives on your workstation to access the data sets. (Use different drive letters than you are already using on your workstation or to access servers in the LAN.) Each of these drive specifications has four parameters: • The drive letter

- The high-level qualifier, most often your user ID, or the high-level qualifier where the project resides on the host
- Whether the data is text or binary
- The mapping for file extensions.

You should have two different drive statements to allow your workstation to access both text and binary files. An example is

```
drive m: cobrs08 text   mapping-parameter
drive n: cobrs08 binary local
```

where the mapping parameter varies between

- local, used when the host mapping is not supported, and
- filesys userid.nfs.mapping, used when the host supports the file name mapping.

If you create the nfs.mapping file as it is described in the next section, the drive m: command looks like this:

```
drive m: cobrs08 text filesys cobrs08.nfs.mapping
```

That means you use the n-drive for binary access and the m-drive for text. The file name mapping is important only for text files. Our COBOL sources for the first sample project are contained in a data set called *COBRS08.HELLO.COBOL*, your job control languages (JCLs) are contained in a data set called *COBRS08.HELLO.JCL*. We have access to both using the m-drive because both data sets are located under our user ID and that is the one we want to access. Our mapping files are contained in a data set called *COBRS08.NFS.MAPPING*, and our sigyclst files will be contained in a data set called *COBRS08.REMOTE.SIGYCLST*. We allocate this data set later. At this moment, we need only know under which high-level qualifier we want to locate it.

All needed data sets are located under the same user ID so we will get them all with this specification.

With these settings, we get access to a drive (n:) that is to be used by the system to access sigyclst files, and one drive (m:) that you use to access COBOL sources and JCL sources.

type

For local mapping, you must specify the substitution for data sets on the host and file extensions on the workstation. This is important only when your NFS on the host does not support file-name mapping. You can leave it as it is when the host NFS includes this function. The file mapping on the host is explained in 2.3, "Configuring the Host for TCP/IP and File Mapping" on page 21. Otherwise, specify the mappings as shown below. For example, members in data sets with the qualifier ′COBOL′ or ′SOURCE′ should be resolved with the extension ′.CBL′ on the workstation. To do so, specify these substitutions in the following way:

```
type  COBOL   CBL
type  SOURCE  CBL
```

If you want to follow our sample application, make sure you have the following substitution commands here:

```
type  COBOL   CBL
type  JCL     JCL
type  COPY    CPY
```

Type all the mappings in this way:

```
type HostType WorkstationExtension
```

where HostType is the type of the data set on the host that contains the members you want to map, and WorkstationExtension is the extension that you get for these files on the workstation.

That does not mean that you get a file extension for every file in the WorkFrame on the PC. You still see the member names without any extension on WorkFrame, but WorkFrame knows what kind of files they are and can exclude inappropriate functionality for files. If you have specified a remote mapping, you do not need to specify these mappings.

localcopy	No action required. This word is reserved for future use.
mvsedit	No action required.
joblog	No action required.
sigyclst	Specify here the name of the sigyclst data set on MVS. Our sigyclst file will be the data set COBRS08.REMOTE.SIGYCLST on the host, so you insert the line `SIGYCLST COBRS08.REMOTE.SIGYCLST`
sysproc	No action required.
tempdrive	No action required.
fsstartcmd	No action required.
fsstopcmd	No action required.
mountcmd	No action required.
umountcmd	No action required.
testfile	Put a comment on this line to inactivate it.
protsave	No action required.
closefile	No action required. But for more information refer to the box called *Protected saving*.
mvscomm	No action required.
header	No action required.
maxcmd	No action required.
nullstdin	No action required.

After changing all the parameters, save this file and close it. Setting up this MVSINFO.DAT file completes configuration of the workstation to use the remote edit, compile, and debug feature.

If NFS on the host does not support Mapping

It is very important that you know whether your NFS supports file name mapping on the host or not.

If your NFS on the host does not support file name mapping, you must make special modifications in the MVSINFO.DAT file on your workstation. Your WorkFrame is able to undertake the mapping. You have to specify the mappings in the MVSINFO.DAT file. But this kind of mapping differs from the mapping that is supported by the host, because it offers you the submit option for every file on the WorkFrame, no matter whether it makes sense or not. After you start the submission for a file that is not JCL, you get the error message ′Action not valid for this part...′. That means the WorkFrame can check the kind of the file only after the submit was performed. If the host NFS supports the name mapping, you do not even get the choice to select the submission for specific files.

Protected saving

If you use only the WorkFrame to edit files, you always get the editor of the host that is called *MVSEDIT* on the workstation to edit your program. This editor provides a protected saving in the way that the file is saved in a temporary file first and then overwritten on the original file. When the allocated space for this data set on the host is no longer sufficient for saving the changed file, you still have the temporary file, which is deleted only after a successful save.

You have the choice to use ′LXPM′ from an OS/2 window also, but then you should be aware that you do not get protected saving. If the space allocated for the data set is insufficient, the data is lost because no temporary file was saved before.

We strongly recommend that you work with the editor provided by MVS. To work with this MVS Editor, specify the close-file parameter in the MVSINFO.DAT file. You must specify

```
closefile yes
```

or

```
closefile no
```

Which you use depends on whether your MVS NFS supports the immediate closing of files or not. If it does, specify yes; it is the fastest way to save your files on the host. Otherwise, specify no, and when you send the temporary file it will be closed after the timeout value has expired. If the MVS NFS does not support the immediate close of files and you specify yes, you get an error message, because the host tries to access a file that is still open.

Also, you can modify the parameter for writetimeout to perform a faster save. Depending on the response time of your MVS system, you can reduce the writetimeout parameter. The better your response time, more you can reduce this parameter. We have set it to 10. The default value for writetimeout is 30.

2.2 Accessing the Host from the Workstation

First we want to connect to the host to check if it works properly.

Once you have updated the MVSINFO.DAT file correctly, you need only type `MVSSTART` in an OS/2 window. This reads the necessary information from the MVSINFO.DAT file, starts NFS, and establishes the connection to the specified MVS system.

To check if the installations and configurations are done correctly, you can set up this command manually in an OS/2 window. For that, type `MVSSTART` in an OS/2 window. The first note you get does not belong to your system. Ignore this message and press Enter. The following messages disappear so quickly on your OS/2 window that you will probably not be able to read them.

The process starts with an NFSSTART and lists all your local drives. Then NFSCLEAN runs to disconnect from all drives that are not local. Because this is your first logon, NFSCLEAN probably does not find any connected drive. If not, you get ′umount′ messages for all remote drives you are connected to.

Next, the NFSSTART command is executed, which returns this error message:

```
NFSSTART: Couldn't open 'D:\MPTN\ETC\fstab': errno=10
Mounting FSTAB entries...
Finished mounting FSTAB entries
```

The path shown in the message refers to the path where the TCP/IP is installed on your workstation. This message has no effect on your connection to the host. It is caused by configuring your NFS according to our instructions. A mount command is missing here, but we find it easier to modify the mount commands in the MVSINFO.DAT file than to keep changing the TCP/IP configuration. Our mount commands belong to the remote edit, compile, and debug function of VisualAge for COBOL.

Next, the mount commands you specified in the MVSINFO.DAT file as drives are performed. All given attributes are listed and the mount command as you have to specify it in the NFS configuration in the TCP/IP configuration notebook is arranged and executed.

An example of the mount of drive M is as follows:

```
Drive M: being accessed
File system = NFS
System = WTSC47
high-level Qualifier = COBRS08
Readtimeout = 10
Attrtimeout = 10
Writetimeout = 10
Usage = text (translation between EBCDIC and ASCII)
Name mapping = local using MVSINFO.DAT file

mount -lCOBRS08 -p******** M:  WTSC47:COBRS08,text,crlf,
nofastfilesize,readtimeout(10),attrtimeout(10),writetimeout(10)
mount: WTSC47:COBRS08,text,crlf,nofastfilesize,readtimeout(10),
attrtimeout(10),writetimeout(10)
```

The number of drives you specified in the MVSINFO.DAT file is the number of mount commands performed.

After successful mounts, the NFS will test the connection to MVS first by trying to get time information from MVS. It then tries to open a specific data set on MVS with the high-level qualifier 'SYS1' that exists in any case. When this is successful, you get the message

```
Connection to MVS host seems OK
```

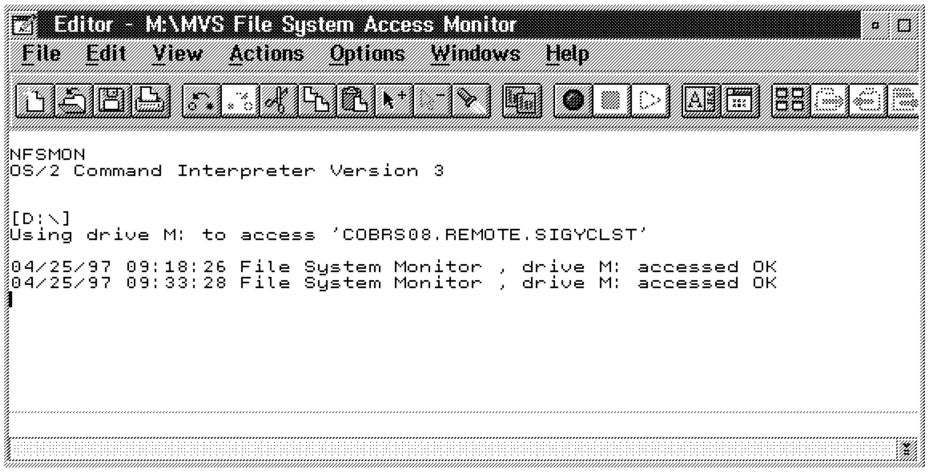
and you can now access the mounted data sets in the same way as you access drives on your workstation or on the LAN on an OS/2 window.

In the meantime, the NFS Control Program window has appeared in the background and listed some information about the connection. This information is of little interest for you, but be aware that you stop NFS when you close this window by double-clicking on the icon in the upper left corner. Shrink the window, but do not close it at this point.

NFS Control Program window

Once you cancel NFS by pressing the Ctrl and C keys together, this window becomes a normal OS/2 window, although the title of the window is still NFS Control Program, implying that it is still active when it is not. The window has no effect on any further activities.

Once the connection is established, an editor comes up in the background and reports that the access to drive M: is OK (Figure 9).



```
NFSMON
OS/2 Command Interpreter Version 3

[D:\]
Using drive M: to access 'COBRS08.REMOTE.SIGYCLST'

04/25/97 09:18:26 File System Monitor , drive M: accessed OK
04/25/97 09:33:28 File System Monitor , drive M: accessed OK
```

Figure 9. Editor – M:\MVS File System Access Monitor Window

The connection to drive m: is checked by NFS every 15 minutes, NFS and this monitor reports the result.

While the mount is still active, you also can get a list of all the mounted drives when you open the window for the drives in OS/2 as shown in Figure 10.

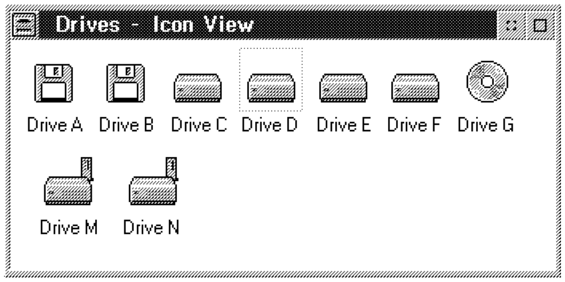


Figure 10. Drives – Icon View Window

Error messages after mounting a drive

When you get an error message like this:

```
mount: writetimeout(10) not in hosts database
Access of drive M: failed
```

after you mounted a drive, it is likely that your user ID on the host does not have full authority for all functions supported by NFS. Depending on the error message, put the referred command in the MVSINFO.DAT file in comments. In our example, put an asterisk at the beginning of the line of the writetimeout command. The error message means that the writetimeout command is not supported in your environment so you cannot use it. For that, the host system takes the defaults and does not accept any entry from you.

Try to mount again without specifying the writetimeout command and you will mount the drive successfully.

To disconnect the host, type `MVSSTOP` in an OS/2 window on a local drive and the unmount progress starts and disconnects you from the host. The following output is produced in the OS/2 window where you set up the command:

```
umount M:
Unmounting ′WTSC47:COBRS08,text,crlf,fileextmap,
sidefile(COBRS08.NFS.MAPPING),nofastfilesize,readtimeout(10),
attrtimeout(10),writetimeout(10)′...      successful.

umount N:
Unmounting ′WTSC47:COBRS08,binary,nofastfilesize,readtimeout(10),
attrtimeout(10),writetimeout(10)′...      successful.

nfsclean
Type Name FSDName FSAData
Local C: FAT
Local D: FAT
Local E: FAT
Local F: UNKNOWN
Local G: UNKNOWN

You should select ″View->Refresh now″ in order to update your
project view

Press any key when ready . . .
```

NFSSTART

If you are interested in starting NFS separately, to test the connection to the host and to see if you can access the data sets, you can start NFS manually and activate the mount.

For that, open an OS/2 window and follow the instructions:

1. Type `NFSSTART` into an OS/2 window. The NFS starts running.
2. Type `mount -luserid -ppassword drive: MVSsystem:hlq,text,crlf` as one command where
   - `userid` is your user ID in the host system you want to access
   - `password` is the password that belongs to this user ID
   - `drive` is any drive letter that you do not use at this moment for any other drive
   - `MVSsystem` is the host system you want to connect
   - `hlq` is a high-level qualifier on this host. If you are not sure if you have authorization for other high-level qualifier use your own user ID as hlq first. That cannot encounter a problem.
3. Check that the mount command is done properly.

Now you can access the specified data set on the host remotely from your workstation.

If MVS logs you off

An MVS system may periodically log off a user. If MVS logs you off, reissue the **umount** and **mount** commands to log on again.

If you are working with a WorkFrame project, select **Project** in the menu bar and then first **Disconnect MVS drives**, and then **Connect MVS drives** to unmount the drives and mount them again. After that press **F5** to refresh the files in the WorkFrame.

In Section 2.3, "Configuring the Host for TCP/IP and File Mapping" we explain how to configure the host to complete your environment.

2.3 Configuring the Host for TCP/IP and File Mapping

To support communication with your workstation, you have to make some changes on the host. For remote edit, compile, and debug, some restructured executor program (REXX) procedures shipped with VisualAge for COBOL have to be uploaded to the host and one of them has to be modified.

Follow these instructions to make your host ready for remote edit, compile, and debug:

1. Create a data set that contains MVS REXX procedures that are used by remote edit, compile, and debug MVS processing. For that, you have to set up your own SIGYCLST data set. To allocate a data set with the low-level qualifier SIGYCLST with the appropriate specifications, make sure that you are connected to the host, and then set up the mount command as follows:

   ```
   md "remote.sigyclst,dsntype(pds),lrecl(255),blksize(6160),recfm(vb)"
   ```

   Or use the emulation to allocate the new data set with these specifications:

Space units	BLOCK
Primary quantity	72
Secondary quantity	16
Directory blocks	7
Record format	VB
Record length	255
Block size	6160

Then upload all files contained in the \IBMCOBOL\CLISTS directory for OS/2 into this SIGYCLST data set. You have three different choices for uploading the files: copy them remotely, use the emulator session for copying it, or use FTP that belongs to TCP/IP. We describe all three kinds of transferring files:

a. To make it easier, go to this drive in an OS/2 window while typing

```
cd \IBMCOBOL\CLISTS
```

and copy all files into the drive you have mounted with file extension mapping. Following our example, copy them to drive m:. Type in this window the commands:

```
copy igyfcmh.cmd m:\remote.sigyclst
copy igyfproh.cmd m:\remote.sigyclst
```

and so on until you have copied all files from the directory \IBMCOBOL\CLISTS to the host data set.

b. You can also upload the files using the emulator session.

c. Another way of uploading the files is to use FTP. This is much easier than using the transfer feature offered by the emulator session, and it is recommended when the copying the files does not work.

You find a detailed explanation of the upload via FTP in 2.4, “Upload Workstation Files Using FTP” on page 25.

Error message while copying files

When you get the error message ′SYS0206: The file name or extension is too long.′ while you copy files from local drives to remote drives, it means that you did not mount the remote drive with file extension mapping. The extension of the file is not accepted by the remote drive. To avoid this error, use a remote drive that has been mounted with file extension mapping.

2. Update the supplied REXX procedure named IGYFLIBS, contained in this data set, to indicate the fully qualified name of the load library. Edit the member IGYFLIBS and change the dsname into the data set name of the load library you want to use. In our case, we changed this command into

```
dsname = ″COBRS08.REMOTE.LOAD″;
```

3. Next, assemble the IGYFINF program and link the object code into the load library specified. For that, up-load the file called IGYFINF.ASM located in the /IBMCOBOL/MACROS directory of your workstation into a data set for assembler programs. Assemble this program using the assembler on MVS and link the object code into the load library you have specified in the step

above. In our environment, the jobs for the compilation of assembler programs are included in the data set SYS1.PROCLIB.

4. If you are using host name mapping, you must specify how the MVS files are to be interpreted based on the MVS data set name low-level qualifier. That means you have to create a library that tells the host NFS how it should replace the data set names with file extensions. To do so, allocate a library that is called *USERID.NFS.MAPPING* where userid is your user ID on this host.

   Either you use the usual way to allocate the library (with the emulator session), or you try it remotely as described below. Using the emulator is probably easier, so you can allocate the library with the following specifications:

Space units	BLOCK
Primary quantity	1
Secondary quantity	1
Directory blocks	1
Record format	FB
Record length	80
Block size	400

   Then, edit the member as you are used to doing it on the host and insert the code shown in Figure 11 on page 24.

   To allocate the library remotely on the host, make sure that you are connected to the host. Then, mount a new drive (one you are not yet using) with the following command in an OS/2 window in one line:

   ```
   mount drive: hostname:″userid,parameters″
   ```

   where

   - Drive is the drive you want to use for the mount
   - Hostname is the alias for the host you want to access
   - Userid is your user ID or another high-level qualifier where the library should reside
   - Parameters is a list of parameters to specify the attributes of the data set you want to allocate.

   To allocate the NFS.MAPPING library, specify the mount with the following parameters:

   ```
   mount o: wtsc47:″cobrs08,lrecl(80),blksize(400),recfm(fb),dsorg(ps)″
   ```

   where o: is the drive we are using for the mount, wtsc47 is the host we are accessing, cobrs08 is our user ID, and the other parameters are for the allocation of the new library NFS.MAPPING. After you have set up the command, you are prompted to type your user ID and your password. You should then get the message that the NFS drive was attached successfully.

   After you allocate the data set, you can specify the name mappings as in the MVSINFO.DAT file for local mappings. For example, you can specify that library names ending with the qualifier COBOL contain COBOL source code. That would mean that every member belonging to a data set with the low-level qualifier COBOL gets the extension .CBL, no matter what the high-level qualifier is.

To edit this file, change to an OS/2 window and enter the name of the mounted drive. In our example it is drive o:, so edit

```
o:
```

in the OS/2 window and then edit the NFS.MAPPING file while typing

```
e NFS.MAPPING
```

The editor comes up and you can edit the code shown in Figure 11.

```
#NFS.MAPPING
**.COBOL      .CBL
**.COPY       .CPY
**.COBCOPY    .CPY
**.OBJ        .OBJ
**.LOAD       .EXE
**.CLIST      .CMD
**.SIGYCLST   .CMD
**.CNTL       .JCL
**.JCL        .JCL
**.LISTING    .LST
**.OUTLIST    .OUT
```

Figure 11. NFS.MAPPING File

To save it, select **File** in the menu bar and then choose **Save**. The file is being saved on the host. Then press F3 to leave the file.

5. After this library creation, unmount the drive o: again, because you do not need it any more with these file specifications. Change the drive in your OS/2 window to any local drive and type

```
umount o:
```

to set the drive o: free.

NFS.MAPPING

In the NFS.MAPPING file, many low-level qualifiers can have the same interpretation as, for example, both the low-level qualifiers COPY and COBCOPY can contain copy books for COBOL programs. All files that share the same low-level qualifier are interpreted the same way. This means every file in a data set with the low-level qualifier COBOL gets the extension .CBL on the workstation.

The effect of this name mapping is that the WorkFrame offers you only the functions whose names fit the file. How an MVS file is interpreted determines whether certain actions are performed for the file. For example, you can only submit jobs from files that are interpreted to contain MVS JCL.

You have now prepared both the workstation and your host environment for using the remote edit, compile, and debug feature and you can start working remotely.

At this point, make sure that the system administrator of the host system has set up the IP address of your workstation for accessing the TCP/IP on the host. The TCP/IP on the host needs to know which user ID accesses the host on TCP/IP from which workstation IP address. If you are using two different workstations, the system administrator must define these two IP addresses for your user ID on the host to give you TCP/IP access.

Section 2.4, “Upload Workstation Files Using FTP” explains the upload of files from the workstation to the host using FTP. You do not have to read it necessarily. It is an easy way to copy files to the host. If you copied your files another way, go directly to Chapter 4, “Creating Applications Using Remote E/C/D on OS/2” on page 59 to create your first application remotely. (You can use FTP to upload the sample application files.)

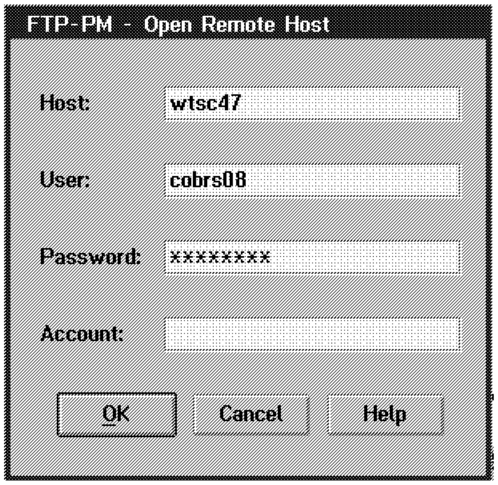
2.4 Upload Workstation Files Using FTP

This section offers you a smart way of uploading files from the workstation or a diskette to the host.

To use FTP, follow these steps:

1. Open the OS/2 System folder located on the Desktop.
2. Open the TCP/IP folder located in the OS/2 System folder.
3. Double-click on the FTP-PM icon to start FTP.

   The **FTP-PM - Open Remote Host** window appears. Specify the host you want to connect to in the *Host:* field, give the user ID you have on this host in the *User:* field, and the password that belongs to this user ID in the *Password:* field (Figure 12) and click on **OK**



Host: wtsc47

User: cobrs08

Password: xxxxxxxx

Account:

Figure 12. FTP-PM – Open Remote Host Window

4. The FTP-PM - hostname (user ID) window comes up and shows you files and directories from your workstation in the upper half of the window and the data set list with your user ID as high-level qualifier in the lower half of the window.

   To upload the CLIST files you need in the SIGYCLST data set on the host, you first have to specify the directories where these files are located.

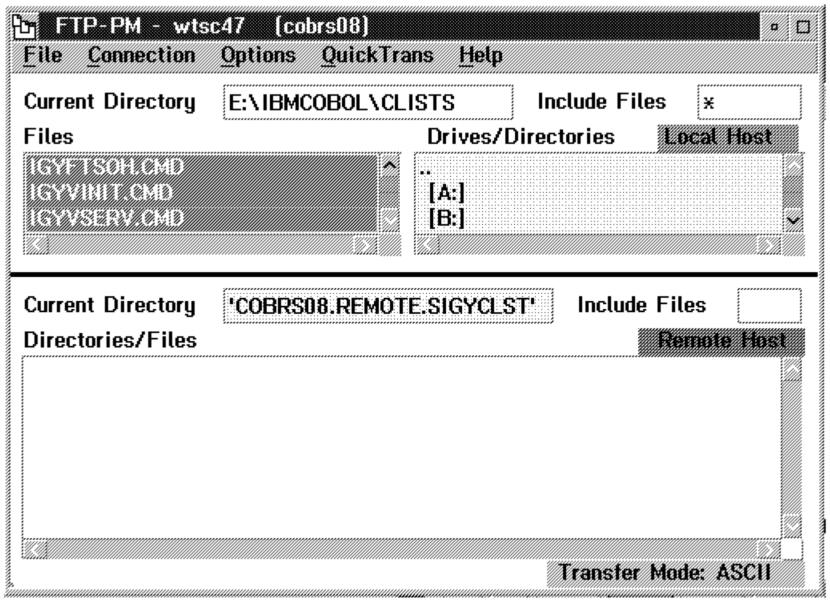
   To reach the directories, specify the drive in the *Current Directory* field where VisualAge for COBOL is installed and press Enter. The contents of the *Files* and the *Drives/Directories* lists change accordingly.

Select **IBMCOBOL** in the *Drives/Directories* list and then scroll down in the list and select **CLISTS**.

The *Files* list shows the files you have to upload.

5. Now turn to the lower half of the window where your host data sets are displayed. Scroll down in this list until you reach the data set you have allocated for the SIGYCLST members. If you followed the instructions, you should find the data set named REMOTE.SIGYCLST. Double-click on it to open this data set.

6. Turn back to the upper half of the window and select all files listed in the *Files* list (Figure 13) and select **QuickTrans** in the menu bar.



Current Directory E:\IBMCOBOL\CLISTS Include Files *

Files: IGYFTSOH.CMD, IGYVINIT.CMD, IGYVSERV.CMD

Drives/Directories: .., [A:], [B:]

Current Directory 'COBRS08.REMOTE.SIGYCLST' Include Files

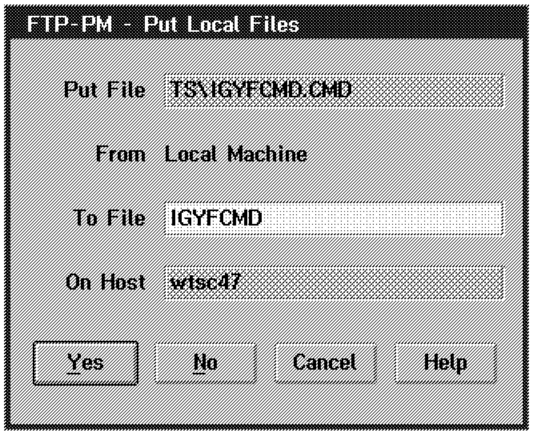
Directories/Files

Transfer Mode: ASCII

Figure 13. FTP-PM - wtsc47 (cobrs08) Window

7. The **FTP-PM - Put Local Files** window appears and refers to the first file of the file list you selected. In the *To File* field, specify the name of this file on the host. Leave the name as is, but remove the extension .CMD as shown in Figure 14 on page 27.

   Click on **Yes** to start the transfer of this file.


FTP-PM - Put Local Files

Put File TS\IGYFCMD.CMD

From Local Machine

To File IGYFCMD

On Host wtsc47

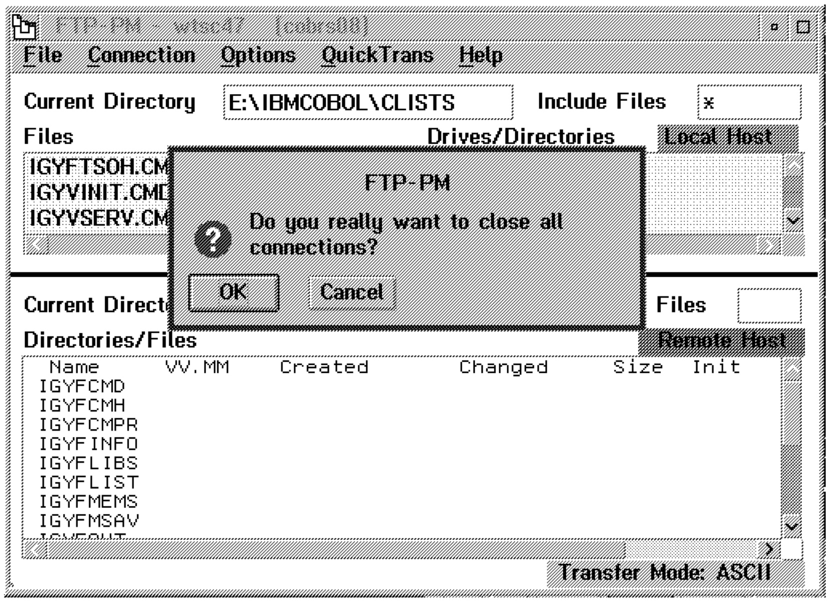
Figure 14. FTP-PM - Put Local Files Window

8. The **FTP-PM - Transfer Progress** window comes up, reports the file transfer and disappears after the transfer has finished.

   The **FTP-PM - Put Local Files** window appears again with the next file. Proceed with all other files in the same way and remove the file extension .CMD from the *To File* entry field.

9. After the last file transfer, the **FTP-PM - wtsc47 (cobrs08)** window gets active again and now shows the transferred files as members in the *Directories/Files* list in the lower half of the window that belongs to the host.

10. Select **Connection** in the menu bar and then **Close all hosts** in the pull-down menu to disconnect the host. The **FTP-PM** window appears and asks you to confirm this request (Figure 15). Click on **Yes**.


FTP-PM - wtsc47 [cobrs08]

Current Directory E:\IBMCOBOL\CLISTS Include Files *

Files Drives/Directories Local Host

IGYFTSOH.CM
IGYVINIT.CMD
IGYVSERV.CM

FTP-PM

Do you really want to close all connections?

Current Direct Files

Directories/Files Remote Host

```
 Name      VV.MM    Created        Changed     Size  Init
IGYFCMD
IGYFCMH
IGYFCMPR
IGYFINFO
IGYFLIBS
IGYFLIST
IGYFMEMS
IGYFMSAV
```

Transfer Mode: ASCII

Figure 15. FTP-PM Window

11. Finally, close the **FTP-PM - wtsc47 (cobrs08)** window (which changed its name to FTP-PM after disconnection) by double-clicking on the icon in its upper left corner.

The file transfer is done. Now you can use the files on the host.

Chapter 3. Remote Edit, Compile, and Debug with APPC on OS/2

To use the protocol APPC to communicate with the host from your workstation, you must set up the configuration on the host and on your workstation. This chapter explains the configurations you need on the workstation and on the host to use APPC and SMARTdata Utilities.

For the requirements on the host side, refer to the Getting Started online manual that is shipped with VisualAge for COBOL.

For communications between the workstation and the host system using APPC plus the SMARTdata Utilities (SdU) component on the workstation, you need an SdU server on the host system. Use DFSMS Release 1.3 or later.

Also you need IBM COBOL for MVS and VM, Version 1.2 or later.

We next explain some products we are using and then in Section 3.1, “Configuring Communications for APPC” on page 30, we describe step by step how you have to configure your communications for APPC on the workstation.

Advanced Program-to-Program Communication (APPC) has been designed to provide enhanced system network architecture (SNA) support for distributed processing. By distributed processing, we mean having two or more processors communicate in the execution of a unit of processing. Such a unit is generally referred to as a *transaction*, so that the term cooperative (or distributed) transaction processing describes the environment addressed by APPC.

Specifically, APPC describes the functions that can be used by programs in separate processors to communicate with each other in the execution of a single distributed transaction. The goal of APPC is to facilitate the development of distributed applications by providing a set of defined functions that serve as the base for program-to-program communication, independent of the types of processors in which those programs run.

A new logical unit (LU) was developed to support distributed applications. This was LU6.2; it was an evolution from LU6.0 (CICS program-to-program) and LU6.1 (CICS and IMS program-to-program). However, as the name ″LU6.2″ was not very meaningful, the term *APPC* came to be used. The two terms, APPC and LU6.2 are synonymous.

The benefit of using APPC is that the programs can be distributed across several processors and can work together cooperatively, using a set of standard conversational verbs to perform work. Also, neither program has to know anything about the environment of the other processor; it only has to be able to hold a conversation with the other program.

IBM SMARTdata UTILITIES is designed to provide local and remote access to data. It offers

- Record-oriented file access through standard COBOL I/O statements to:
  - Local OS/2 VSAM files
  - Remote MVS VSAM, SAM, PDS, and PDSE files
  - Remote OS/400 Record Files
  - Remote CICS-managed VSAM files on MVS using CICS/DDM

- A full set of data conversion APIs for converting single, double, and mixed-byte character strings, numerics and complex structured records.
- A full set of SMARTsort APIs for sorting, copying, and merging record and byte files located locally or remotely.

In other words, SdU provides a local record-level access method using VSAM for the workstation. This allows you to have sequential, direct, and keyed files on your local system. Also, it provides you with 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.

SdU further offers a general-purpose data conversion engine for more complex conversion tasks.

And finally, SMARTsort is an industrial-strength sort, merge, copy, and extract package.

DFSMS/MVS provides storage management, data access, device support, program management, and distributed data access for the MVS/ESA platform and participates in the OS/390 Operating System and SystemView for MVS solution set.

Significant improvements are introduced in the following areas:

- Space allocation and extension outage (X37) reduction
- DFSMShsm processing (DFSMShsm CDS access, DFSMShsm Duplex Tape)
- VSAM (VSAM key sequenced data set loading and buffering options)
- Catalog (Catalog Search Interface)
- Compression
- Tape processing
- New statistics records that will aid in reporting, analysis and planning.

Additional items will aid in productivity, distributed data processing with a distributed file manager (DFM) Data Agent, and system-managed storage extensions to support the parallel Sysplex.

3.1 Configuring Communications for APPC

Before you start with any configuration, make sure that your workstation fulfills the software requirements for the use of the remote edit, compile, and debug component. Check that the operating system is IBM OS/2 Warp Version 3.0 or later, including FixPak 26 or IBM OS/2 Warp Version 4.0 including FixPak 1.

Further, make sure that at least the following VisualAge for COBOL components are installed:

- Remote edit, compile, and debug
- Editor
- WorkFrame.

In the following, we explain how to set up your workstation for communication with the host by way of APPC. Configuring APPC cooperative sessions requires coordination between workstation and host administrators. You need some information from your system administrator—as, for example, the network ID.

Configuring Communications Manager/2 at the workstation requires:

- Defining the workstation to the network. Define a link from the workstation to the host, or to an intermediate APPN node.
- Defining the workstation as a client for APPC sessions. Define the workstation as a server for remote debug APPC sessions.

In addition to configuring Communications Manager/2 for OS/2 Warp, you need to do some SMARTdata Utilities configuration.

3.1.1 Installing Communications Manager/2 for OS/2 Warp

For the use of APPC, you need to install Communications Manager/2 for OS/2 Warp to set up independent logical units to establish a link between workstation and mainframe. That enables you to connect to the host through APPC.

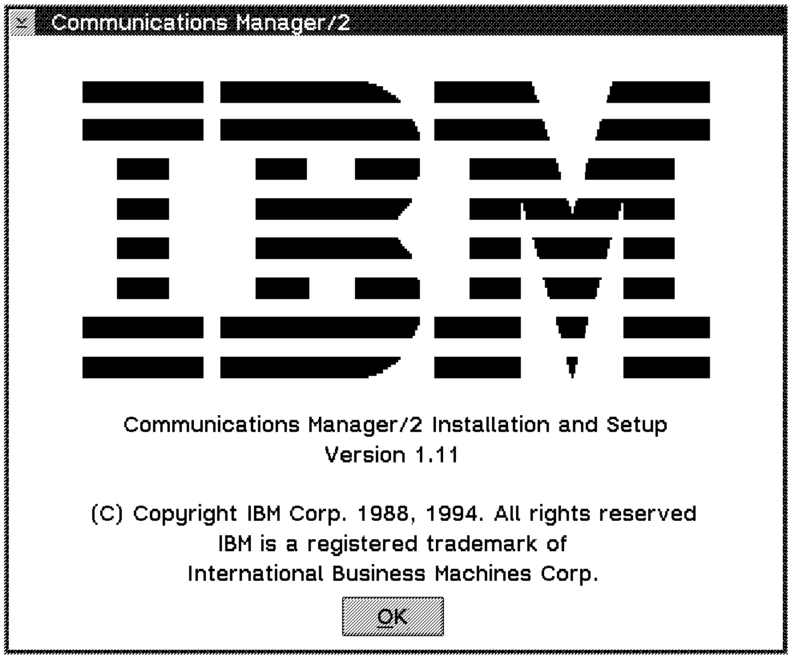
First, install Communications Manager/2 for OS/2 Warp on your workstation, following these steps:

1. Insert the Communications Manager CDROM in the CDROM drive. At an OS/2 command prompt, type

   ```
   D:\CM2\CMSETUP
   ```

   where D is the drive letter of the CDROM, and press Enter.

2. Click on the **OK** push button on the Installation of Communications Manager window (Figure 16).



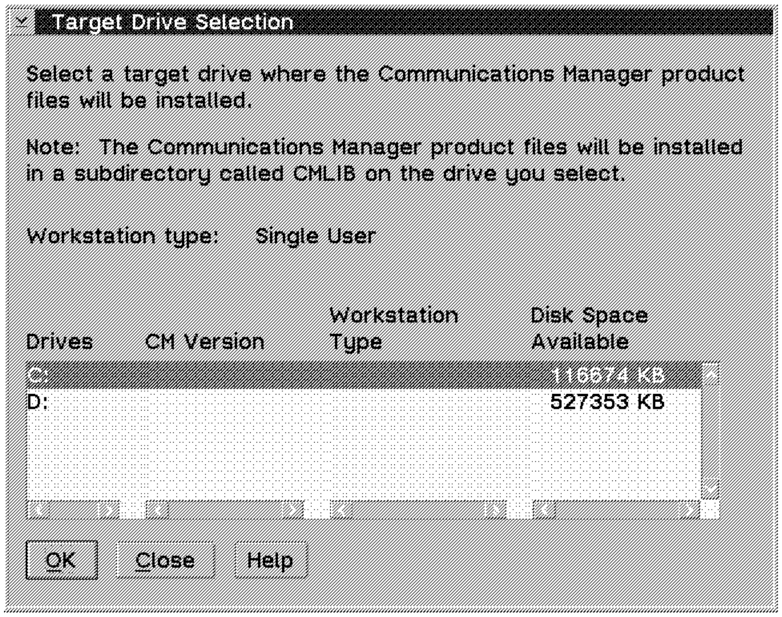
Communications Manager/2 Installation and Setup
Version 1.11

(C) Copyright IBM Corp. 1988, 1994. All rights reserved
IBM is a registered trademark of
International Business Machines Corp.

Figure 16. Installation Window for Communications Manager 2

3. The **Installation Notes** window appears. Click on **Continue**.

4. On the **Target Drive Selection** window (Figure 17 on page 32) select the drive where you want to install Communications Manager/2 for OS/2 Warp and click on **OK**.



Select a target drive where the Communications Manager product files will be installed.

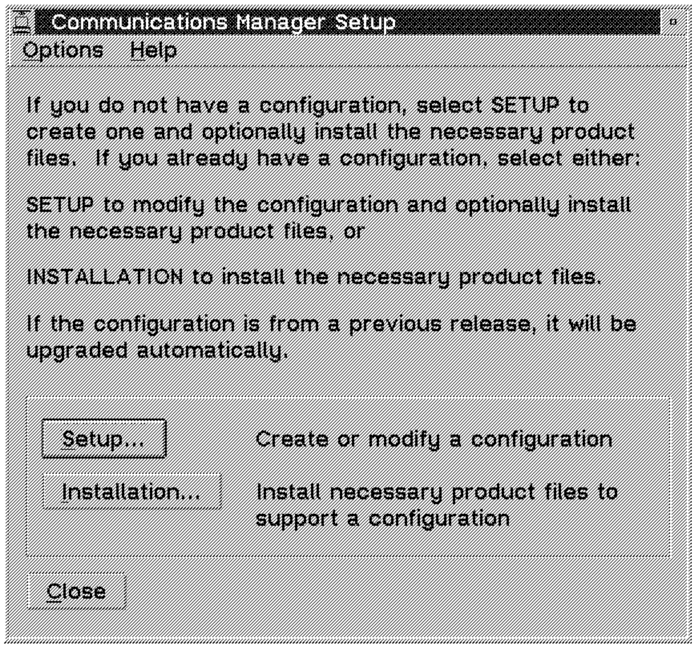
Note: The Communications Manager product files will be installed in a subdirectory called CMLIB on the drive you select.

Workstation type: Single User

Drives	CM Version	Workstation Type	Disk Space Available
C:			116674 KB
D:			527353 KB

Figure 17. Target Drive Selection Window

5. On the **Communications Manager Setup** window (Figure 18) select **Setup...** to configure the machine.



If you do not have a configuration, select SETUP to create one and optionally install the necessary product files. If you already have a configuration, select either:

SETUP to modify the configuration and optionally install the necessary product files, or

INSTALLATION to install the necessary product files.

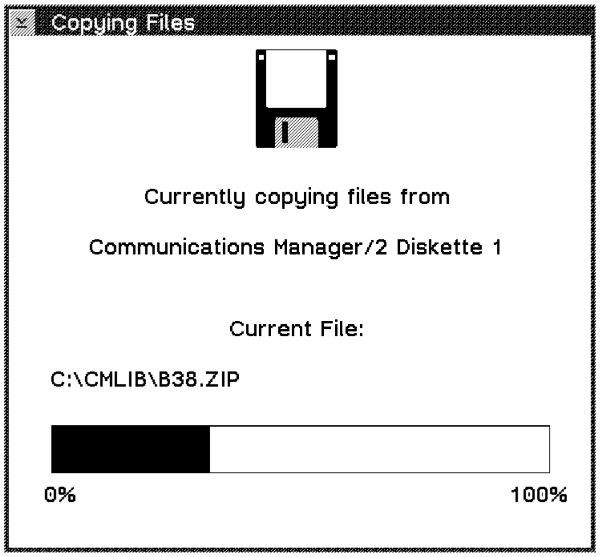
If the configuration is from a previous release, it will be upgraded automatically.

Setup... Create or modify a configuration

Installation... Install necessary product files to support a configuration

Figure 18. Communications Manager Setup Window

6. While the system copies files from the CDROM, it shows in the **Copying Files** window that it still works (Figure 19 on page 33).



Copying Files

Currently copying files from

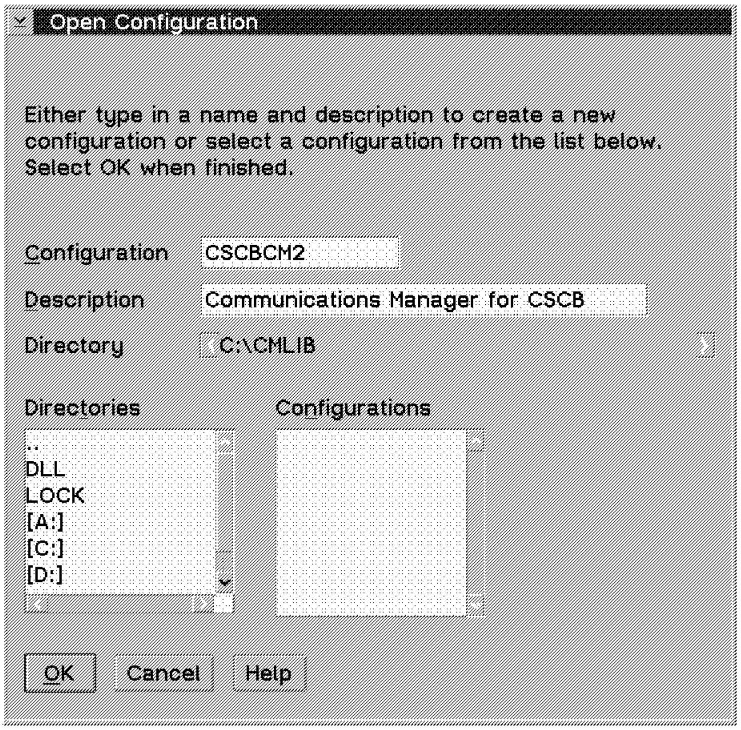
Communications Manager/2 Diskette 1

Current File:

C:\CMLIB\B38.ZIP

Figure 19. Copying Files Window

7. The **Open Configuration** window appears (Figure 20). Type a name for this configuration in the appropriate field (a description for the configuration is optional) and click on **OK**.



Open Configuration

Either type in a name and description to create a new configuration or select a configuration from the list below. Select OK when finished.

Configuration: CSCBCM2

Description: Communications Manager for CSCB

Directory: C:\CMLIB

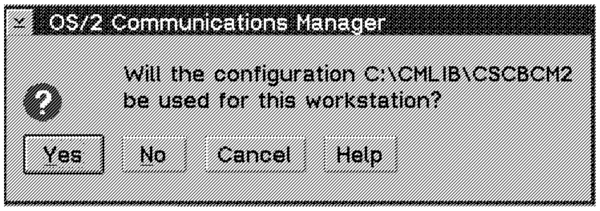
Directories:
..
DLL
LOCK
[A:]
[C:]
[D:]

Configurations:

Figure 20. Open Configuration Window

8. On the **OS/2 Communications Manager** window, click on **Yes** to create the new configuration.

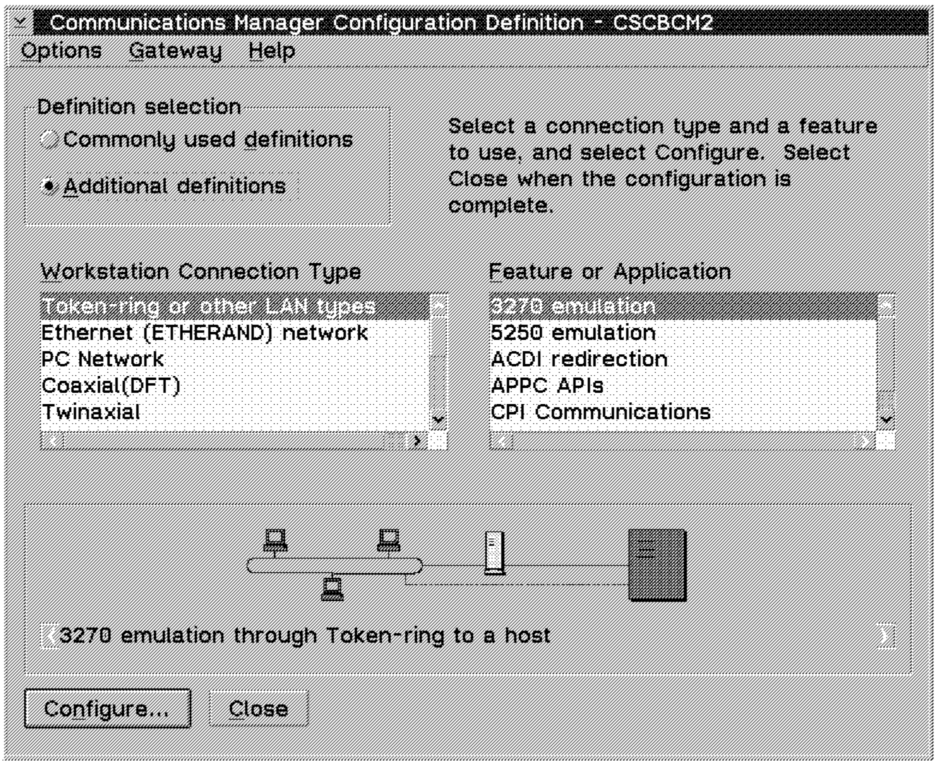
9. On the next **OS/2 Communications Manager** window (Figure 21 on page 34), click on **Yes** to use the configuration for this workstation.



Will the configuration C:\CMLIB\CSCBCM2 be used for this workstation?

Figure 21. OS/2 Communications Manager Window—Workstation

10. On the **Communications Manager Configuration Definition** window (Figure 22) you have to define the connection types you want to use. In the field *Additional definitions* select **Token-ring or other LAN types** as the workstation connection type and **3270 emulation** as the feature for the application. Then click on **Configure...**.


Communications Manager Configuration Definition - CSCBCM2
Options Gateway Help
Definition selection
Commonly used definitions
Additional definitions
Workstation Connection Type
Token-ring or other LAN types
Ethernet (ETHERAND) network
PC Network
Coaxial(DFT)
Twinaxial
Feature or Application
3270 emulation
5250 emulation
ACDI redirection
APPC APIs
CPI Communications
3270 emulation through Token-ring to a host
Configure...
Close

Select a connection type and a feature to use, and select Configure. Select Close when the configuration is complete.

Figure 22. Communications Manager Configuration Definition Window

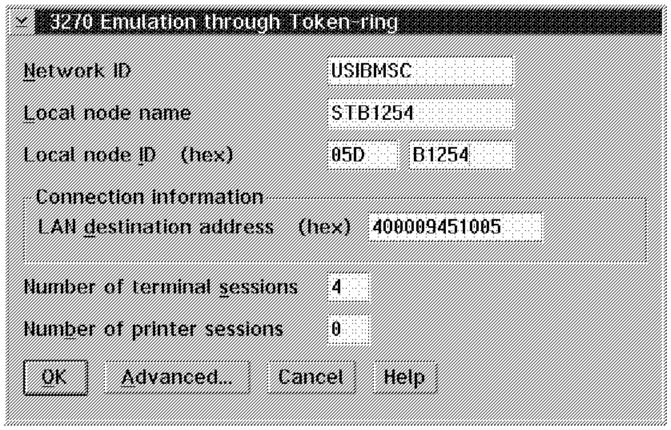
11. The **3270 Emulation through Token-ring** window appears (Figure 23 on page 35). You have to specify the following parameter for the emulation sessions:

   - Network ID is the name of the network where your workstation is located.
   - Local node name is the name of the workstation as it is known on the network. You are creating a new configuration, so type the name to be assigned to your local node (your workstation). This name becomes the control-point name for your node.
   - Local node ID is made up of the characters that form the last eight digits used in the exchange identification (XID) for activating a link. The first

three characters default to X′05D′. Accept this default. The coordinator of the host computer you connect to can tell you which local node ID to use.

- The LAN destination address is the address of the adapter on your network′s communications controller or gateway. For a 3270 emulation configuration, the LAN destination address is the address of the network adapter for your SNA gateway, or your SNA controller.
- The number of terminal sessions can be between one and four.
- Finally, specify the number of printer sessions you want to configure.

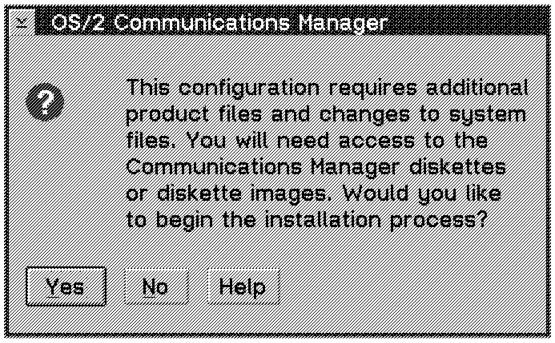
Click on **OK**.



Network ID: USIBMSC
Local node name: STB1254
Local node ID (hex): 05D B1254
Connection information
LAN destination address (hex): 400009451005
Number of terminal sessions: 4
Number of printer sessions: 0

Figure 23. 3270 Emulation through Token-ring Window

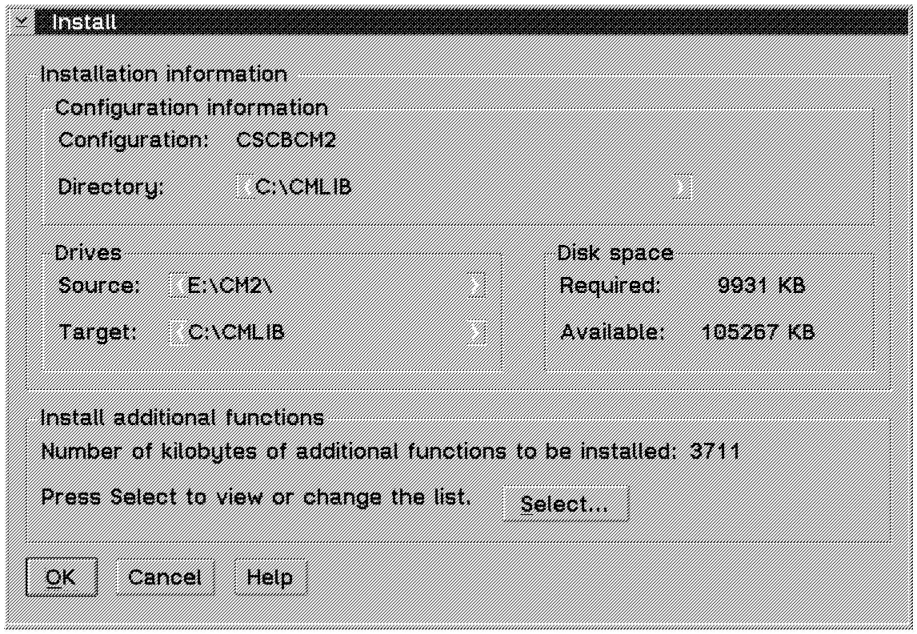
12. After the set up of the 3270 emulation, click on **Yes** on the window labeled **OS/2 Communications Manager** (Figure 24).



This configuration requires additional product files and changes to system files. You will need access to the Communications Manager diskettes or diskette images. Would you like to begin the installation process?

Figure 24. OS/2 Communications Manager Window—Product Files

13. On the **Install** window, click on **OK** (Figure 25 on page 36).


Install

Installation information

Configuration information

Configuration: CSCBCM2

Directory: C:\CMLIB

Drives

Source: E:\CM2\

Target: C:\CMLIB

Disk space

Required: 9931 KB

Available: 105267 KB

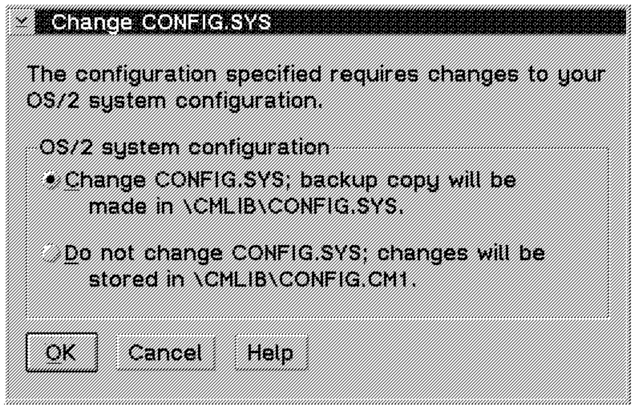
Install additional functions

Number of kilobytes of additional functions to be installed: 3711

Press Select to view or change the list.

Figure 25. Install Window—Communications Manager/2 for OS/2 Warp

14. On the **Change CONFIG.SYS** window, accept the default and click on **OK** (Figure 26).


Change CONFIG.SYS

The configuration specified requires changes to your OS/2 system configuration.

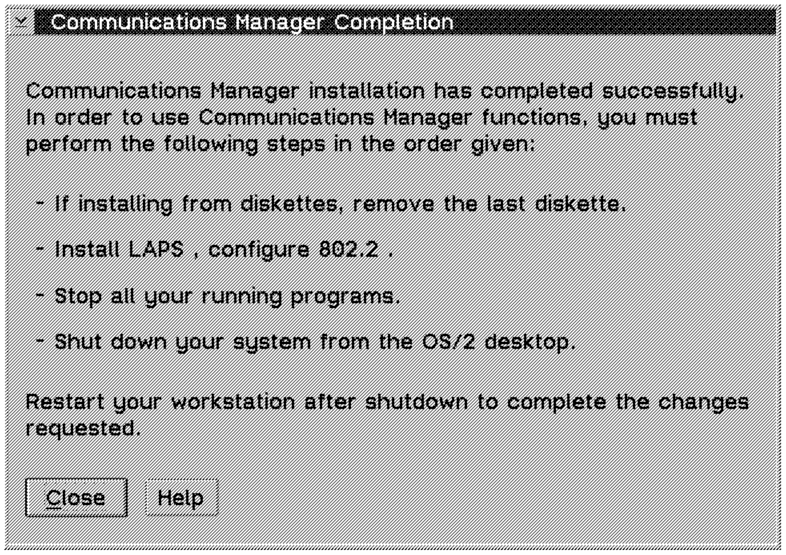
OS/2 system configuration

● Change CONFIG.SYS; backup copy will be made in \CMLIB\CONFIG.SYS.

○ Do not change CONFIG.SYS; changes will be stored in \CMLIB\CONFIG.CM1.

Figure 26. Change CONFIG.SYS Window

15. Close the configuration by clicking on **Close** in the **Communications Manager Completion** window (Figure 27 on page 37).



Communications Manager Completion

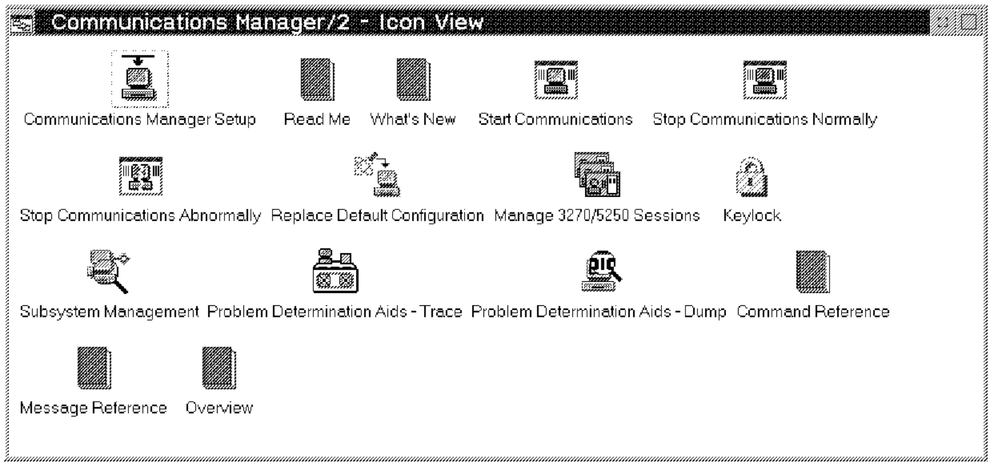
Communications Manager installation has completed successfully. In order to use Communications Manager functions, you must perform the following steps in the order given:

- If installing from diskettes, remove the last diskette.
- Install LAPS , configure 802.2 .
- Stop all your running programs.
- Shut down your system from the OS/2 desktop.

Restart your workstation after shutdown to complete the changes requested.

Figure 27. Communications Manager Completion Window

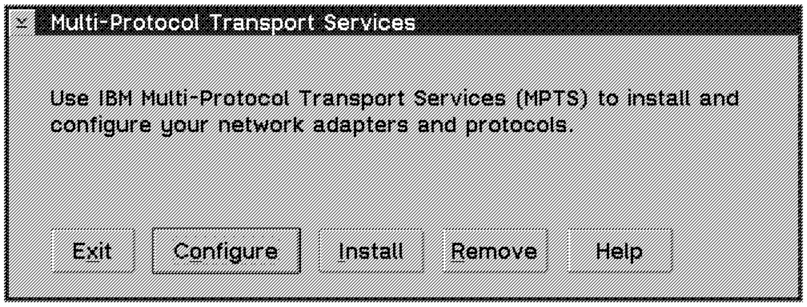
16. You have now finished the installation of Communications Manager/2 for OS/2 Warp. You can find the Communications Manager/2 icon on your Desktop. Double-click on this icon to open the **Communications Manager/2 - Icon View** window (Figure 28).



Communications Manager/2 - Icon View
Communications Manager Setup
Read Me
What's New
Start Communications
Stop Communications Normally
Stop Communications Abnormally
Replace Default Configuration
Manage 3270/5250 Sessions
Keylock
Subsystem Management
Problem Determination Aids - Trace
Problem Determination Aids - Dump
Command Reference
Message Reference
Overview

Figure 28. Communications Manager/2 - Icon View Window

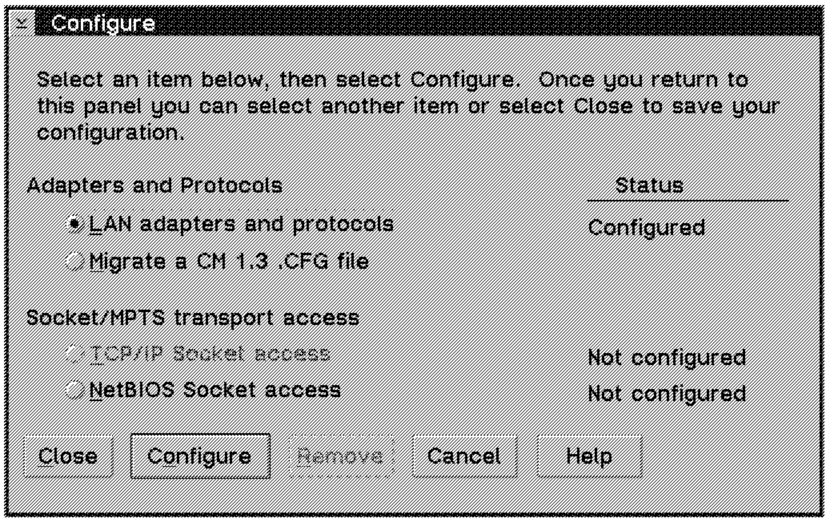
17. Shut down your workstation and reboot. To finish the configuration of Communications Manager/2, double-click on the **MPTS icon** on your desktop. Click on **OK** on the **Multi-Protocol Transport Services-Logo** window, then click on **Configure** on the **Multi-Protocol Transport Services** window (Figure 29 on page 38).


Multi-Protocol Transport Services

Use IBM Multi-Protocol Transport Services (MPTS) to install and configure your network adapters and protocols.

Figure 29. Multi-Protocol Transport Services Window

18. LAN adapters and protocols may already be configured on your machine, but we need to add a protocol. Select **LAN adapters and protocols** and click on **Configure** on the **Configure** window (Figure 30).

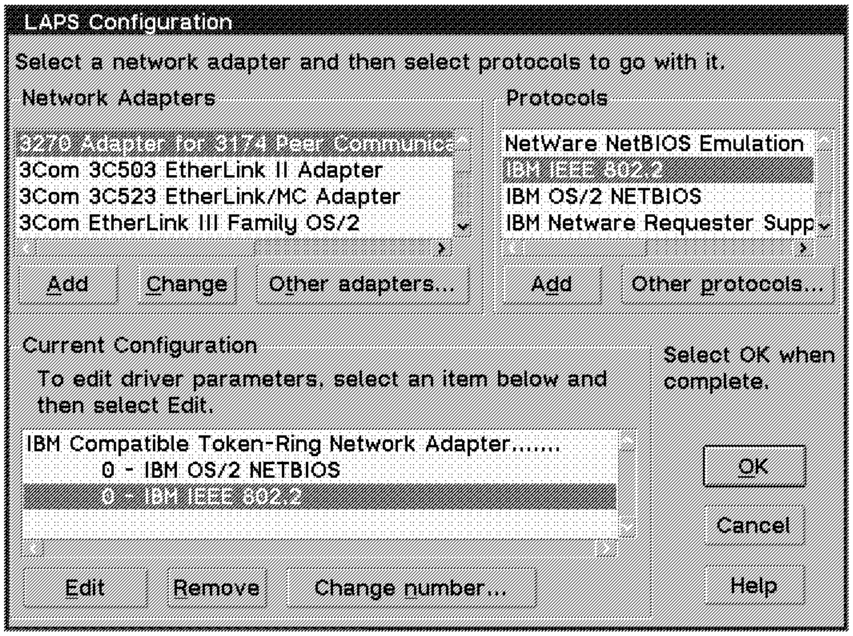

Configure

Select an item below, then select Configure. Once you return to this panel you can select another item or select Close to save your configuration.

Adapters and Protocols	Status
● LAN adapters and protocols	Configured
○ Migrate a CM 1.3 .CFG file	
Socket/MPTS transport access	
○ TCP/IP Socket access	Not configured
○ NetBIOS Socket access	Not configured

Figure 30. Configure Window—MPTS

19. On the **LAPS Configuration** window (Figure 31 on page 39), select the network adapter **3270 Adapter for 3174 Peer Communications** and **IBM IEEE 802.2** as its protocol. LAPS is the LAN adapter and protocol support of multiprotocol transport networking (MPTN). IEEE is the Institute of Electrical and Electronic Engineers. Click on **Add** in the part of the window for the protocols, and IBM IEEE 802.2 appears in the current configuration list. You need not change the default parameters of this protocol. Click on **OK**


LAPS Configuration

Select a network adapter and then select protocols to go with it.

Network Adapters
- 3270 Adapter for 3174 Peer Communica
- 3Com 3C503 EtherLink II Adapter
- 3Com 3C523 EtherLink/MC Adapter
- 3Com EtherLink III Family OS/2

Protocols
- NetWare NetBIOS Emulation
- IBM IEEE 802.2
- IBM OS/2 NETBIOS
- IBM Netware Requester Supp

Current Configuration

To edit driver parameters, select an item below and then select Edit.

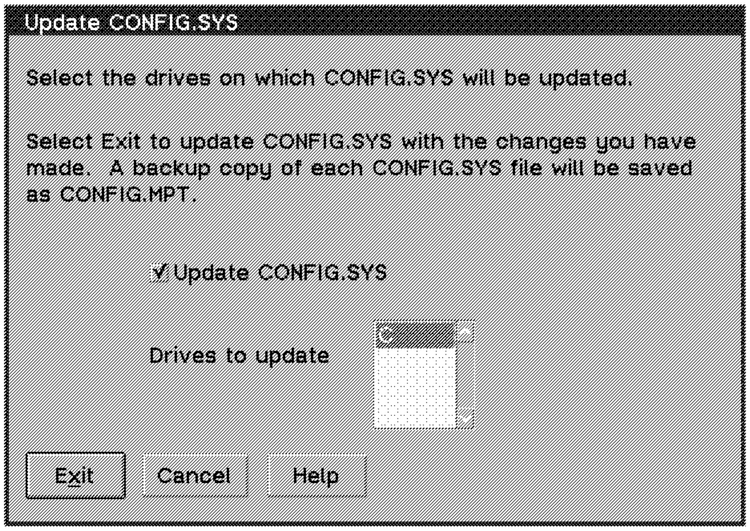
```
IBM Compatible Token-Ring Network Adapter.......
        0 - IBM OS/2 NETBIOS
        0 - IBM IEEE 802.2
```

Select OK when complete.

Figure 31. LAPS Configuration Window

20. Figure 30 on page 38 appears again. Click on **Close** and on **Exit** on the next window.

21. Ensure on the **Update CONFIG.SYS** window (Figure 32) that **update CONFIG.SYS** is selected and click on **Exit**.


Update CONFIG.SYS

Select the drives on which CONFIG.SYS will be updated.

Select Exit to update CONFIG.SYS with the changes you have made. A backup copy of each CONFIG.SYS file will be saved as CONFIG.MPT.

[✓] Update CONFIG.SYS

Drives to update: C

Figure 32. Update CONFIG.SYS Window

22. The message window for Update CONFIG.SYS shows you the successful update of the CONFIG.SYS file. Click on **OK** on this window.

23. To exit MPTS, click on **Exit** on the **Exiting MPTS** window.

24. Shut down your workstation.

With the next start of your workstation, the sessions for the 3270 Emulation will appear.

3.1.2 Configuring Communications Manager/2 for OS/2 Warp

While APPC does not require 3270 emulator sessions at the workstation, we assume that you want to run 3270 emulators in parallel with APPC sessions. After you have done the following configuration, you get four 3270 emulator sessions and two APPC sessions.

Further, we assume that you want to configure APPC for a workstation that is connected to a Token-ring Local Area Network (LAN). That means we are defining the target systems in the Communications Manager configuration files. Distributed FileManager for OS/2 (DFM) uses the SNA LU6.2 protocol for communicating with target systems. You need to have this software installed:

- IBM OS/2 WARP Version 3 plus FixPak 26 or later or IBM OS/2 Warp Version 4 plus FixPak 1
- Communications Manager/2 for OS/2 Warp Version 1.11 or later.
- Distributed FileManager for OS/2.

We define the following parameters in Communications Manager/2 for OS/2 Warp:

- Those that must match parameters of other products:
  - Local node ID
  - Local LU name
  - Mode name
  - Partner LU name
  - Network ID
- Additional parameters:
  - Communication & Systems Management (C&SM) LAN ID
  - Local node name
  - LAN destination address
  - Partner node name
  - Change number of sessions

Follow these steps to set up APPC in Communications Manager/2 for OS/2 Warp:

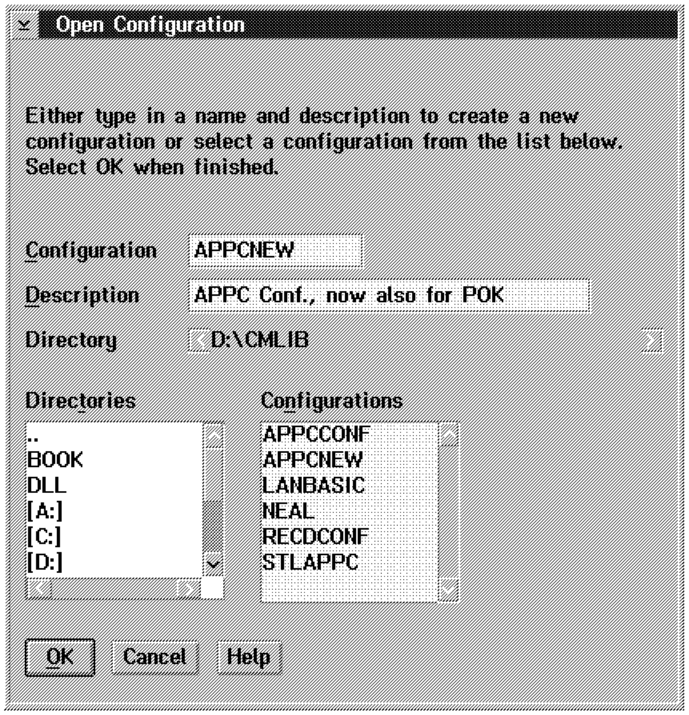
1. On the desktop, double-click on the **Communications Manager/2** icon. The **Communications Manager/2—Icon View** window appears.

   If your Communications Manager is started, stop it by double-clicking on the **Stop Communications Normally** icon.

2. Double-click on the **Communications Manager Setup** icon. Click on **OK** on the **Communications Manager/2** window.

3. The **Communications Manager Setup** window is displayed as in Figure 18 on page 32 in 3.1.1, “Installing Communications Manager/2 for OS/2 Warp” on page 31. Click on **Setup...**.

4. The **Open Configuration** window appears (Figure 33 on page 41). If you installed Communications Manager/2 for OS/2 Warp as described in 3.1.1, “Installing Communications Manager/2 for OS/2 Warp” on page 31, `APPCNEW` is already displayed in the *Configuration* field. Otherwise specify this value (and, optionally, a description) in order to create a new configuration file. Click on **OK**.



Open Configuration

Either type in a name and description to create a new configuration or select a configuration from the list below. Select OK when finished.

Configuration: APPCNEW

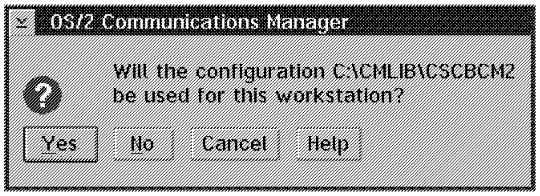
Description: APPC Conf., now also for POK

Directory: D:\CMLIB

Directories	Configurations
..	APPCCONF
BOOK	APPCNEW
DLL	LANBASIC
[A:]	NEAL
[C:]	RECDCONF
[D:]	STLAPPC

Figure 33. Open Configuration Window for APPC

5. The **OS/2 Communications Manager** window is displayed (Figure 34). Click on **Yes**.



OS/2 Communications Manager

Will the configuration C:\CMLIB\CSCBCM2 be used for this workstation?

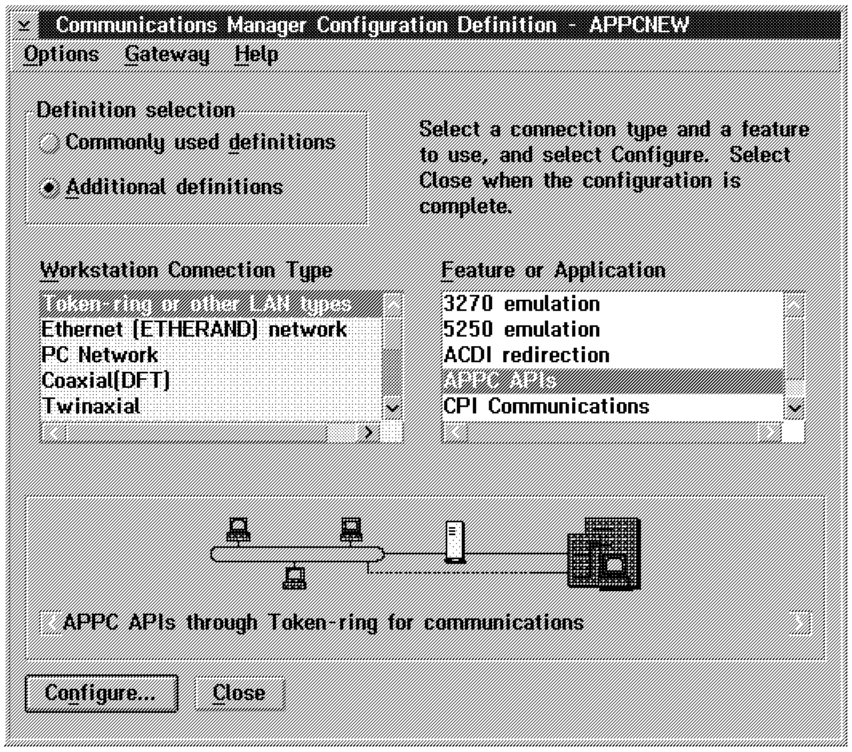
Figure 34. OS/2 Communications Manager Window for APPC

6. The **Communications Manager Configuration Definition—APPCNEW** window appears (Figure 35 on page 42). Ensure that the **Additional definitions** radio button is selected.

   Select **Options** from the menu bar, and **Use advanced configuration** from the first pull-down menu and **On** from the second.

   Select **Token-ring or other LAN types** from the *Workstation Connection Type* list box and **APPC APIs** from the *Feature or Application* list box. The window shows a graphic of APPC APIs through Token-ring for communications (Figure 35 on page 42).

   Click on **Configure...**.


Communications Manager Configuration Definition - APPCNEW

Options Gateway Help

Definition selection

○ Commonly used definitions

● Additional definitions

Select a connection type and a feature to use, and select Configure. Select Close when the configuration is complete.

Workstation Connection Type

- Token-ring or other LAN types
- Ethernet (ETHERAND) network
- PC Network
- Coaxial(DFT)
- Twinaxial

Feature or Application

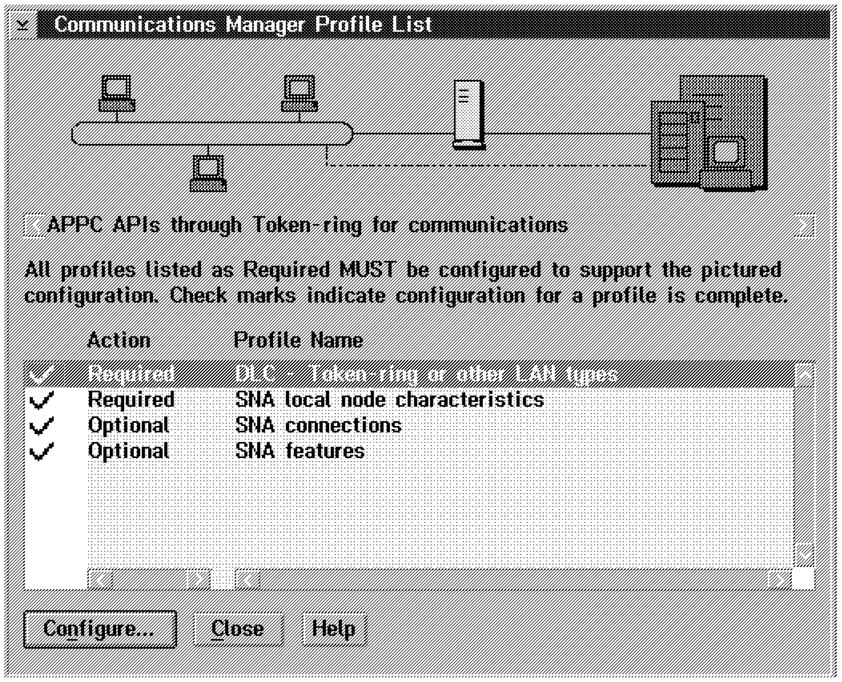
- 3270 emulation
- 5250 emulation
- ACDI redirection
- APPC APIs
- CPI Communications

APPC APIs through Token-ring for communications

Configure... Close

Figure 35. Communications Manager Configuration Definition Window

7. The **Communications Manager Profile List** window appears (Figure 36). Select **DLC—Token-ring or other LAN types** from the *Profile Name* list box and click on **Configure...**. DLC is data link control.


Communications Manager Profile List

APPC APIs through Token-ring for communications

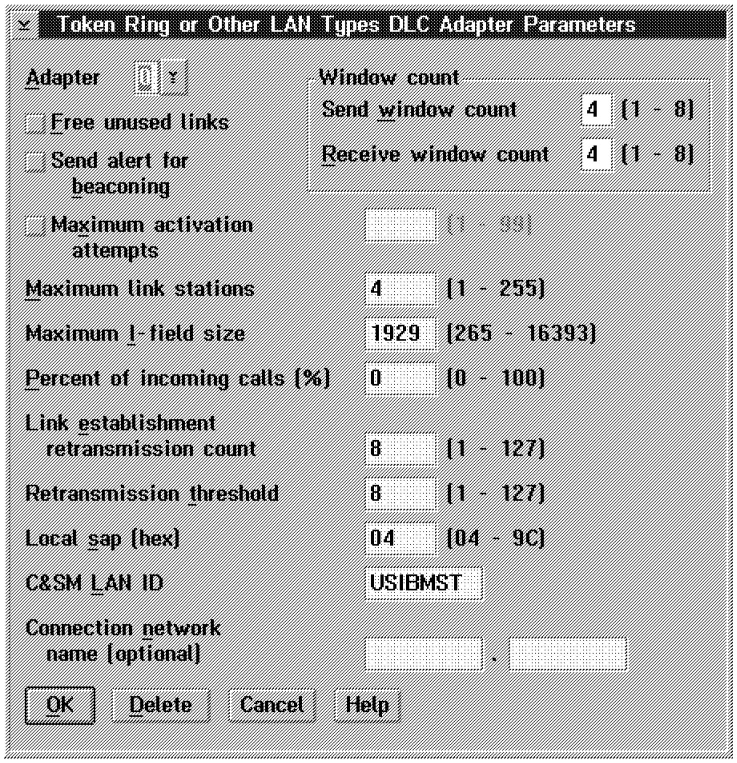
All profiles listed as Required MUST be configured to support the pictured configuration. Check marks indicate configuration for a profile is complete.

	Action	Profile Name
✓	Required	DLC - Token-ring or other LAN types
✓	Required	SNA local node characteristics
✓	Optional	SNA connections
✓	Optional	SNA features

Configure... Close Help

Figure 36. Communications Manager Profile List Window for APPC

8. Type the name of the network where the Token-ring is defined in the *C&SM LAN ID* field of the **Token Ring or Other LAN Types DLC Adapter Parameters** window (Figure 37 on page 43). This field is used for system management and has nothing to do with the APPC client and server set-up. Click on **OK**.



Adapter 0

Window count
Send window count 4 [1 - 8]
Receive window count 4 [1 - 8]

Free unused links
Send alert for beaconing
Maximum activation attempts [1 - 99]

Maximum link stations	4	[1 - 255]
Maximum I-field size	1929	[265 - 16393]
Percent of incoming calls (%)	0	[0 - 100]
Link establishment retransmission count	8	[1 - 127]
Retransmission threshold	8	[1 - 127]
Local sap (hex)	04	[04 - 9C]
C&SM LAN ID	USIBMST	
Connection network name (optional)	.	

Figure 37. Token Ring or Other LAN Types DLC Adapter Parameters Window

9. Select **SNA local node characteristics** from the *Profile Name* list box of the **Communications Manager Profile List** window (Figure 36 on page 42) and click on **Configure...**.

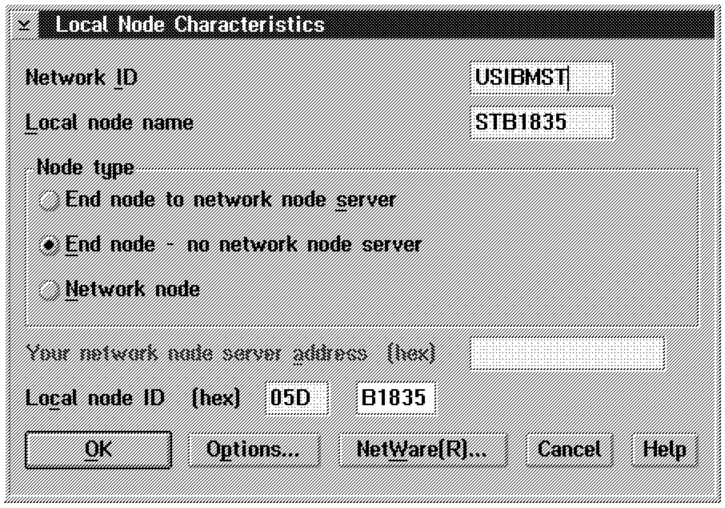
10. The **Local Node Characteristics** window is displayed (Figure 38 on page 44).

    Type your value for the network ID in the *Network ID* field, in our example USIBMST.

    Type a name in the *Local node name* field. The local node name is the name that other nodes in the network use to address this node—for example, in an advanced peer-to-peer network (APPN). For our configuration, this name is not relevant, but it must be unique within the network. You can specify any meaningful name like the virtual telecommunications access method PU name as we used it.

    Type your value for the Local node ID in the *Local node ID* field, in our example 05D B1835.

    Click on **OK**.



Local Node Characteristics

Network ID: USIBMST

Local node name: STB1835

Node type

( ) End node to network node server
(●) End node - no network node server
( ) Network node

Your network node server address (hex)

Local node ID (hex): 05D B1835

Figure 38. Local Node Characteristics Window

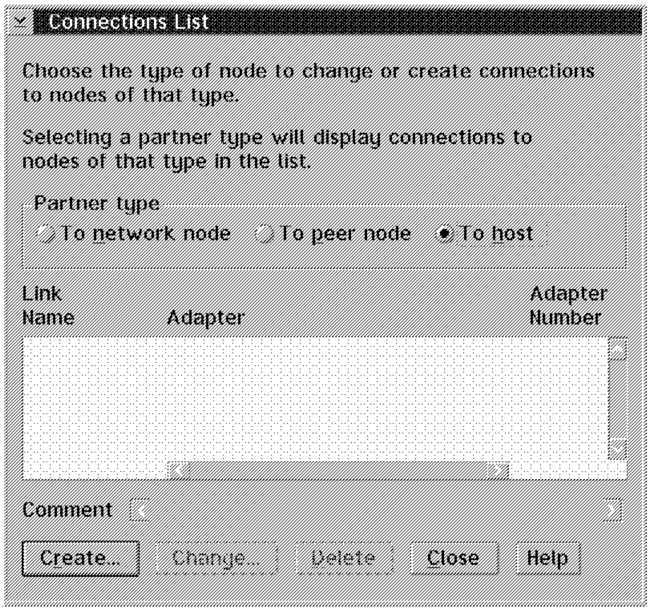
11. Select **SNA connections** from the *Profile Name* list box of the **Communications Manager Profile List** window (Figure 36 on page 42) and click on **Configure...**.

12. The **Connections List** window is displayed (Figure 39).

    Ensure that the **To host** radio button is selected and, if you have already defined a 3270 emulation for the current configuration, the list box on the **Connections List** window shows the following entry:

    ```
    HOST0001    Token-ring or other LAN types     0
    ```

    If this is the case, select this entry and click on **Change...**. Otherwise click on **Create...**.



Connections List

Choose the type of node to change or create connections to nodes of that type.

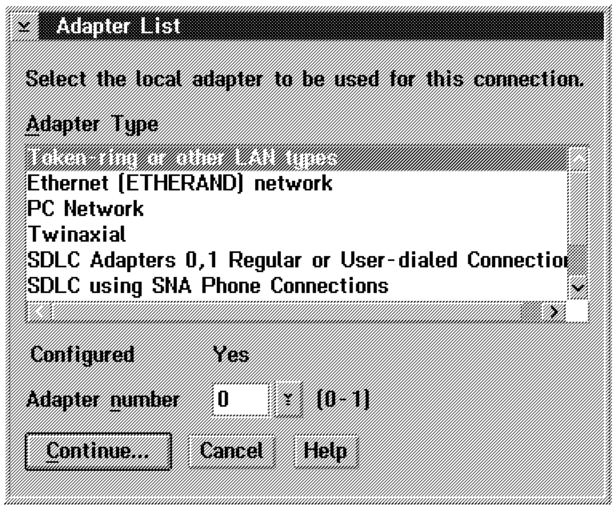
Selecting a partner type will display connections to nodes of that type in the list.

Partner type

( ) To network node ( ) To peer node (●) To host

Figure 39. Connections List Window

13. Select **Token-ring or other LAN types** from the *Adapter Type* list box on the **Adapter List** window (Figure 40 on page 45) and click on **Continue...**.



Select the local adapter to be used for this connection.

Adapter Type

Token-ring or other LAN types
Ethernet (ETHERAND) network
PC Network
Twinaxial
SDLC Adapters 0,1 Regular or User-dialed Connectio
SDLC using SNA Phone Connections

Configured Yes

Adapter number 0 (0-1)

Figure 40. Adapter List Window

14. The **Connection to a Host** window appears (Figure 41 on page 46).

    Depending on whether you are creating a new host link or changing an existing host link, more or fewer fields are already filled.

    Accept the default value for the *Link name* field (HOST0001) or replace it with a more meaningful name for it. The link name is known only to your Communications Manager and is used to identify what parameter definitions belong together.

    The Node ID should already be entered, in our example 05D B1835.

    Enter your LAN destination address in the *LAN destination address* field. In our network this is the Token-ring address of the network controller.

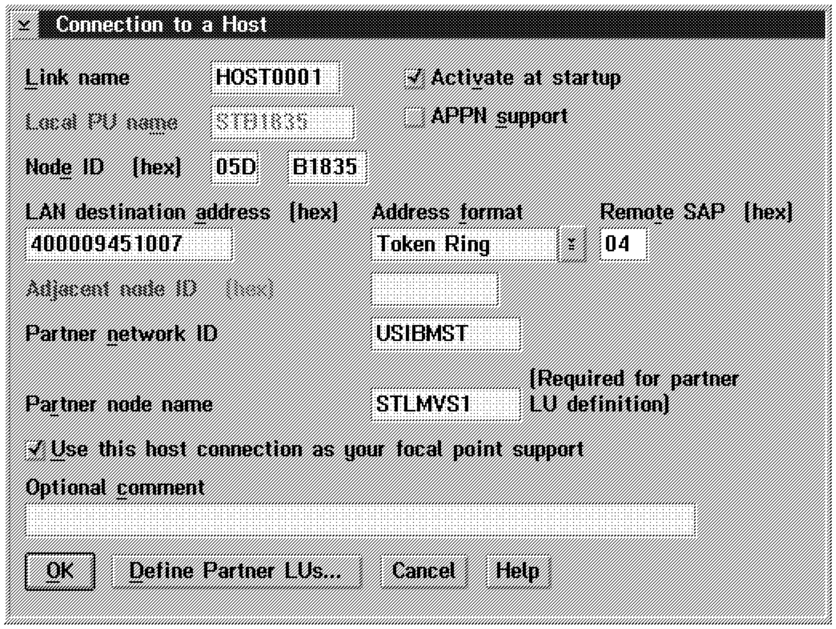
    Type your value for the Network ID in the *Partner network ID* field, in our example USIBMST.

    Type STLMVS1 in the *Partner node name* field.

    You can type any comment in the *Optional comment* field or leave it blank.

    Ensure that the **APPN support** check box is not checked.

    Click on **Define Partner LUs...**.


Connection to a Host

Link name: HOST0001

[✓] Activate at startup

Local PU name: STB1835

[ ] APPN support

Node ID (hex): 05D B1835

LAN destination address (hex): 400009451007

Address format: Token Ring

Remote SAP (hex): 04

Adjacent node ID (hex):

Partner network ID: USIBMST

Partner node name: STLMVS1 (Required for partner LU definition)

[✓] Use this host connection as your focal point support

Optional comment:

Figure 41. Connection to a Host Window

15. The **Partner LUs** window (Figure 42 on page 47) appears.

    Type your value for the network ID in the *Network ID* field—in our example USIBMSC.

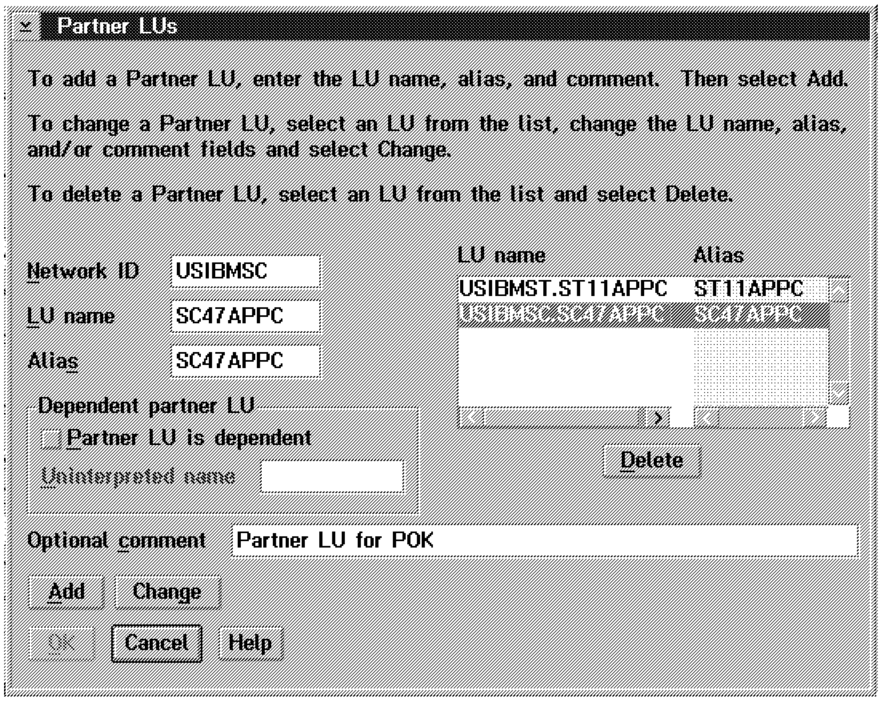
    Type your value for the Partner LU in the *LU name* and in the *Alias* field, in our example SC47APPC. Write this alias in upper case letters.

    You can type any comment in the *Optional comment* field or leave it blank.

    Click on **Add**. The LU name is displayed in the *LU name and Alias* list boxes (Figure 42 on page 47).

    Click on **OK**.

    You can define more than one Partner LU. In our configuration we added a second Partner LU with the network ID USIBMST and the LU name ST11APPC.


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: USIBMSC
LU name: SC47APPC
Alias: SC47APPC

Dependent partner LU
Partner LU is dependent
Uninterpreted name

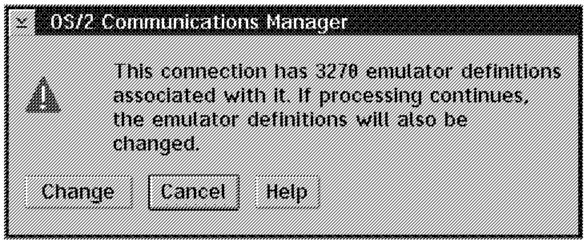
LU name	Alias
USIBMST.ST11APPC	ST11APPC
USIBMSC.SC47APPC	SC47APPC

Optional comment: Partner LU for POK

Figure 42. Partner LUs Window

16. Click on **OK** on the **Connection to a Host** window.

    If the **OS/2 Communications Manager** window appears as in Figure 43, click on **Change**.


OS/2 Communications Manager

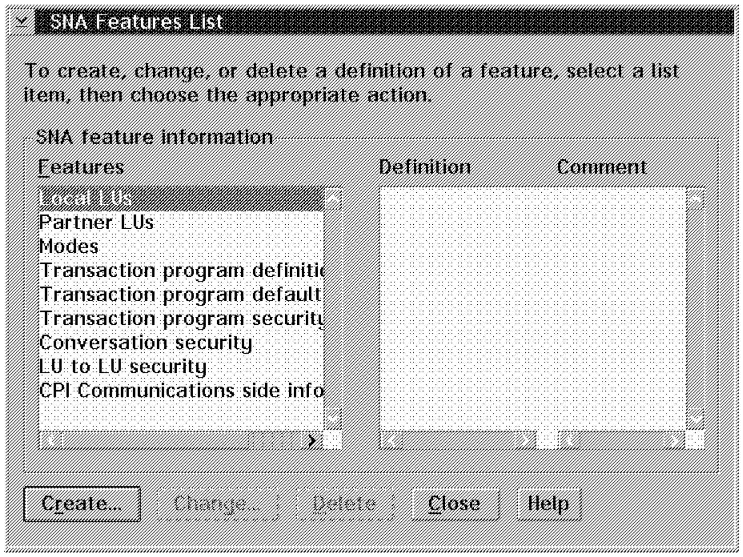
This connection has 3270 emulator definitions associated with it. If processing continues, the emulator definitions will also be changed.

Figure 43. OS/2 Communications Manager Window (3270 emulator information)

Click on **Close** on the **Connection List** window which now shows the following entry in the list box:

```
APPCNEW     Token-ring or other LAN types     0
```

17. Select **SNA features** from the *Profile Name* list box of the **Communications Manager Profile List** window (Figure 36 on page 42) and click on **Configure...**.
18. The **SNA Features List** window is displayed (Figure 44 on page 48). Select **Local LUs** from the *Features* list box and click on **Create...**.


SNA Features List

To create, change, or delete a definition of a feature, select a list item, then choose the appropriate action.

SNA feature information

Features: Local LUs, Partner LUs, Modes, Transaction program definiti, Transaction program default, Transaction program securit, Conversation security, LU to LU security, CPI Communications side info

Definition   Comment

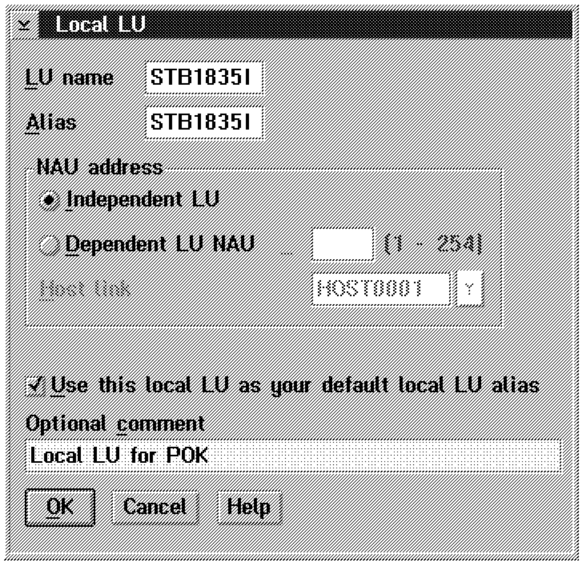
Figure 44. SNA Features List Window

19. The **Local LU** window is displayed (Figure 45).

    Type your value for the Local LU in the *LU name* field and in the *Alias* field—in our example, STB1835I.

    You can type any comment in the *Optional comment* field or leave it blank.

    Click on **OK**.


Local LU

LU name: STB1835I

Alias: STB1835I

NAU address

● Independent LU

○ Dependent LU NAU ... (1 - 254)

Host link: HOST0001

✓ Use this local LU as your default local LU alias

Optional comment: Local LU for POK

Figure 45. Local LU Window

20. The LU name is displayed in the *Definition Comment* list box of the **SNA Features List** window (Figure 45).

    Select **Modes** from the *Features* list box and click on **Create...**.

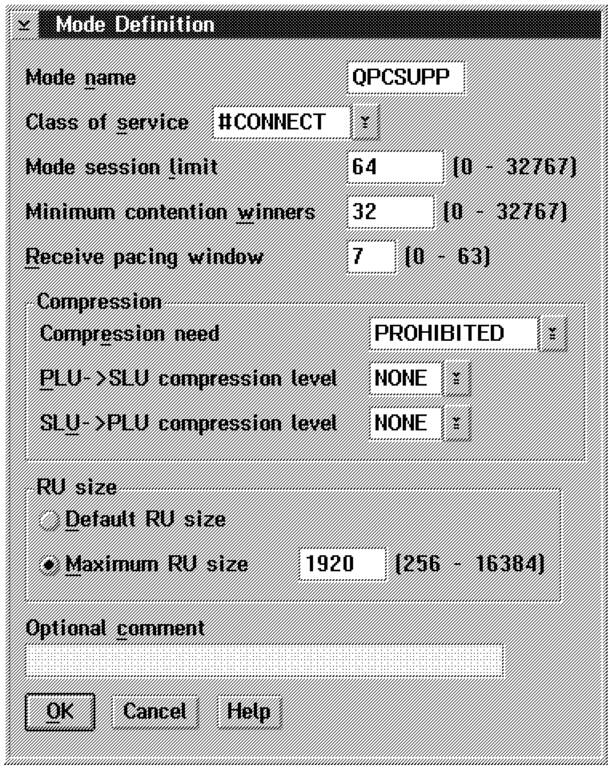
21. The **Mode Definition** window is displayed (Figure 46 on page 49).

Type your value for the Mode name in the *Mode name* field. In our example, the mode name is QPCSUPP.

Select **#CONNECT** from the *Class of service* combination box.

You can type any comment in the *Optional comment* field or leave it blank.

Click on **OK**.



Mode name: QPCSUPP
Class of service: #CONNECT
Mode session limit: 64 [0 - 32767]
Minimum contention winners: 32 [0 - 32767]
Receive pacing window: 7 [0 - 63]
Compression
Compression need: PROHIBITED
PLU->SLU compression level: NONE
SLU->PLU compression level: NONE
RU size
○ Default RU size
● Maximum RU size: 1920 [256 - 16384]
Optional comment:

Figure 46. Mode Definition Window

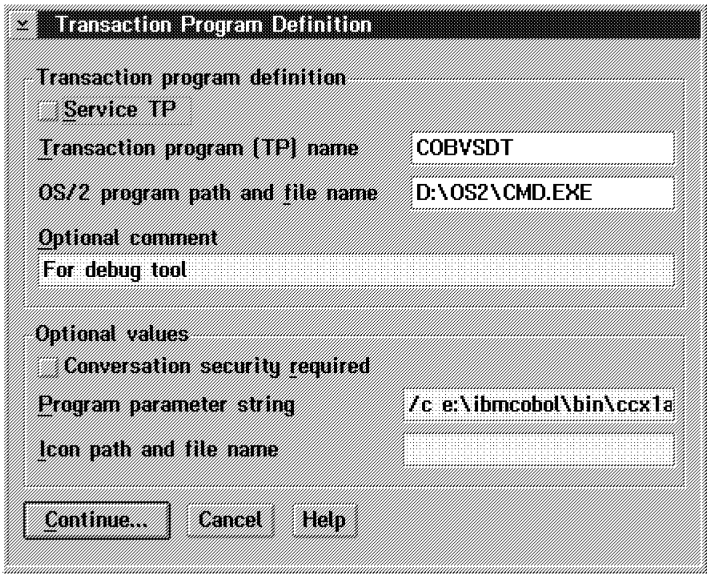
22. Select **Transaction program definitions** from the *Features* list box in the **SNA Feature List** window and click on **Create...**.

    The **Transaction Program Definition** window appears as shown in Figure 47 on page 50.

    Type in the *Transaction program (TP) name* field the value `COBVSDT` and in the field *OS/2 program path and file name* the path `D:\OS2\CMD.EXE` where D: is the drive where OS/2 Warp is installed on your workstation. A comment in the *Optional comment* field is optional.

    In the *Program parameter string* field type the following: `/c E:\IBMCOBOL\BIN\CCX1APCD.CMD` where E: is the drive where your VisualAge for COBOL is installed. Ensure that both the *Service TP* and the *Conversation security required* check-boxes are disabled.

    Select the **Continue** push button.



Transaction Program Definition

Transaction program definition

☐ Service TP

Transaction program (TP) name: COBVSDT

OS/2 program path and file name: D:\OS2\CMD.EXE

Optional comment: For debug tool

Optional values

☐ Conversation security required

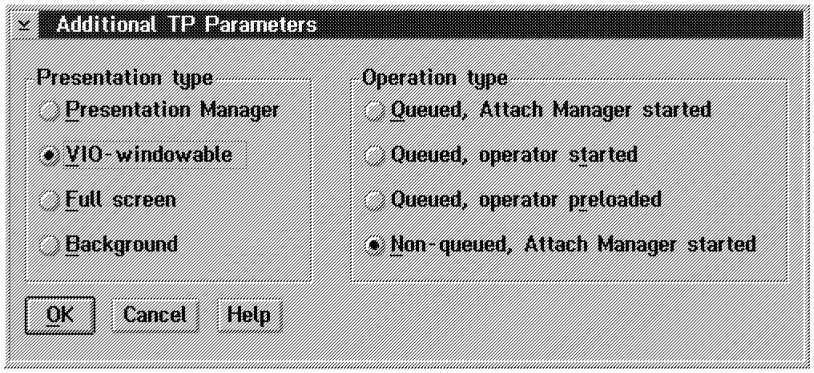
Program parameter string: /c e:\ibmcobol\bin\ccx1a

Icon path and file name:

Figure 47. Transaction Program Definition Window

23. The **Additional TP Parameters** window comes up (Figure 48).

    Click on (enable) the check-box **VIO-windowable** as the *Presentation type* and **Non-queued, Attach Manager started** as the *Operation type*. Then click on **OK**.



Additional TP Parameters

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

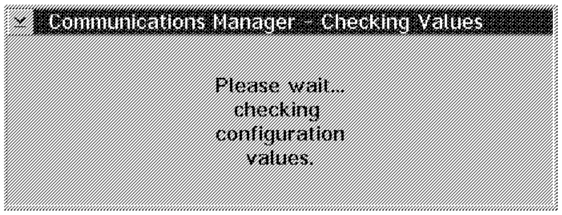
Figure 48. Additional TP Parameters Window

24. Click on **Close** on the **SNA Features List** window.

    Click on **Close** on the **Communications Manager Profile List** window.

    Click on **Close** on the **Communications Manager Configuration Definition—APPCNEW** window.

    The configuration is being validated while the **Communications Manager—Checking Values** window is shown (Figure 49 on page 51).


Communications Manager - Checking Values

Please wait...
checking
configuration
values.

Figure 49. Communications Manager—Checking Values Window

Click on **Close** on the **Communications Manager Setup** window.

3.2 Set up the PC to Connect to the Host

To set up your PC, first you have to put some definitions about your environment in the file called MVSINFO.DAT on your PC. In OS/2 this file is located in the path X:\IBMCOBOL\MACROS where X is the drive on which VisualAge for COBOL is installed.

You find a very detailed description of this file in Section 2.1.2, "MVSINFO.DAT File Explanation" on page 12. The explanations about various functions are the same for both APPC and TCP/IP. If you read the section in TCP/IP, ignore the actions for the TCP/IP connection. Here, we do not describe the reserved words but only the changes you have to make in this file to prepare it for the connection to the host via APPC.

To edit the file, open an OS/2 window and type

```
IWZSET
```

Set all the options that you need to configure the connection as it is described below.

system	No action required.
worksys	No action required. This word is reserved for future use.
userid	Change the name that follows to the user ID you use. We use the ID userid COBRS08.
pwd	Change the name that follows with the encrypted password of the user ID you use for the connection to the host. Instead of the encrypted password, you can specify that you want to be prompted to give the password manually. In this case insert the string pwd ++++++++ 1 instead of typing the encrypted password.
filesys	Specify the system you are using to access the host files. We left the default options filesys sdu accessmon testaccess.
readtimeout	No action required. We put in 10.
writetimeout	No action required. We put in 10.
nfs	No action required.
sdu	You can leave it as is to use the default values. We specified sdu pc_ccsid(850).

drive	This statement allows you to access the data sets on the host. Specify
	1. The drive letter
	2. The high-level qualifier of the data set you want to access
	3. The specification of whether the data format is text or binary
	4. Local mapping for file extensions. Host file mapping is not supported by SMARTdata Utilities, so you have to map the file extensions locally.
	
	You should at least have these two different drive statements about binary and text formatted files:
	`drive m: cobrs08 text   local`
	`drive n: cobrs08 binary local`

type	For local mapping, you must specify the substitution for data sets on the host and file extensions on the workstation. Type all the mappings in this way: `type HostType WorkstationExtension` where the HostType is the type of the data set on the host that contains the members you want to map, and WorkstationExtension is the extension that for these files on the workstation.
localcopy	No action required. This word is reserved for future use.
mvsedit	No action required.
joblog	No action required.
sigyclst	Specify here the name of the SIGYCLST data set on MVS. Our SIGYCLST file is the data set COBRS08.REMOTE.SIGYCLST on the host, so you insert the line `SIGYCLST COBRS08.REMOTE.SIGYCLST`
sysproc	No action required.
tempdrive	No action required.
fsstartcmd	No action required.
fsstopcmd	No action required.
mountcmd	No action required.
umountcmd	No action required.
testfile	Put a comment on this line to inactivate it.
protsave	Type `Yes 1` to ensure getting the protected saving.
closefile	No action required. It belongs to NFS. A change has no effect.
mvscomm	No action required. It also belongs to TCP/IP and NFS.
header	No action required.
maxcmd	No action required.

nullstdin	No action required.

In addition, you should configure the SdU to get the file extension mapping for APPC supported.

3.2.1 Configuration of SMARTdata Utilities on the Workstation

SMARTdata Utilities (SdU) is needed in your environment to support the file extension mapping for APPC.

Distributed FileManager for OS/2 (DFM) belongs to SMARTdata Utilities and is needed for remote record access. It enables an OS/2 Version 2.0 application program to use byte-stream and record-oriented access to remote file data.

Set up the CONFIG.DFM file that is located in the path D:\IBMCOBOL\MACROS where D: is the drive where you have installed VisualAge for COBOL. Change the following variables only:

remote_lu	Type in the alias for the LU name you specified in the Communications Manager configuration in Figure 42 on page 47.
description	The description is optional.
userid	Type in the user ID you are using to access the specified system.
LOCAL_LU	The local LU name is specified in the Communications Manager configuration in Figure 45 on page 48. Give this as the local LU.
MODE_NAME	Type in the mode name you specified in Figure 46 on page 49.
DEFAULT_DFM_TARGET	This should be the same as the DFM target for the remote LU.

Leave the other parameter as is.

A sample of the CONFIG.DFM file is shown below:

```
; Definition of an MVS/ESA system as target:
      DFM_TARGET (
            remote_lu(SC47APPC)
            description(The MVS/ESA System in POK)
            conversation(HOLD)
            max_send_limit(4096)
            userid (COBRS08)
            )
; ******************************************************************
; * Define the local LU alias as defined for the OS/2 ES Communicati
; ******************************************************************
      LOCAL_LU (STB1835I)
; ******************************************************************
; * Define the Mode Name
; ******************************************************************
      MODE_NAME (QPCSUPP)
; ******************************************************************
; * Define the Default DFM Target System
; ******************************************************************
```

```
     DEFAULT_DFM_TARGET(SC47APPC)
; *****************************************************************
; * Define the default CCSID (Coded-Character-Set-Id, up to 5 decimal
; * for all accessed targets
; *****************************************************************
     DEFAULT_CCSID(00850)
; ***************************************************************
; * Define the shared memory size to be available for DFM/2 tracing
; * Trace buffer memory to be specified in kilobytes:
; *  Minimum value:   10 KB
; *  Maximum value: 1000 KB
; ****************************************************************
;      TRACE_BUFFER(64)
; ****************************************************************
; * Define space available for DFM/2 caching.
; * The size (maximum space used) is specified in kilobytes,
; * default value is 2000 KB.
; * The default directory is %ehndir%\cache.
; * Make sure the directory specified exists and does not contain a
; * you want to keep, as DFM/2 may delete all files in that directo
; ****************************************************************
;      DFM_CACHE (
;            cache_disk_size(1000)
;            cache_directory(%ehndir%\cache)
;            )
```

With this, the configuration of SdU is done and you can start configuring your host.

3.3 Configuring the Host for APPC

To support communication with your workstation, you have to make some changes on the host. For remote edit, compile, and debug, some REXX procedures shipped with VisualAge for COBOL must be uploaded to the host and one of them has to be modified.

To make your host ready for remote edit, compile, and debug, you first have to create a data set that contains the MVS REXX procedures to be used. For that, you have to set up your own SIGYCLST data set. Enter the following to allocate a data set with the low-level qualifier SIGYCLST, using the logon at the host in an emulator session, and allocate the data set with these specifications:

Space units	BLOCK
Primary quantity	72
Secondary quantity	16
Directory blocks	7
Record format	VB
Record length	255
Block size	6160

Then upload all files contained in the \IBMCOBOL\CLISTS directory for OS/2 into this SIGYCLST data set.

Error message while copying files

If you get the error message “SYS0206: The file name or extension is too long” while you copy files from local drives to remote drives, then you did not attach the remote drive with file extension mapping. The extension of the file is not accepted by the remote drive. To avoid this error, use a remote drive attached with file extension mapping.

Update the supplied REXX procedure named IGYFLIBS, contained in this data set, to indicate the fully qualified name of the load library. Edit the member IGYFLIBS and change the dsname into the data-set name of the load library you want to use. In our case, we changed this command into

```
dsname = ″COBRS08.REMOTE.LOAD″;
```

Next, assemble the IGYFINF program and link the object code into this specified load library. For that, upload the file called IGYFINF.ASM located in the /IBMCOBOL/MACROS directory of your workstation into a data set for assembler programs. Assemble this program using the assembler on MVS and link the object code into the load library you have already specified. In our environment, the jobs for the compilation of assembler programs are included in the data set SYS1.PROCLIB.

If you have done this, both your host environment and your workstation should be ready for connection and for work with remote edit, compile, and debug.

The host preparations, as described at the beginning of Chapter 3, “Remote Edit, Compile, and Debug with APPC on OS/2” on page 29 must be done by the system administrator. If any connection does not seem to be okay for you, contact your system administrator and ask if the host is set up in complete conformance with the description in this section.

The next step is to test full functionality with the host connected, the drives attached, and the first small application as a starting point.

3.4 Accessing the Host from the Workstation

Once you have updated the MVSINFO.DAT file correctly, you only need to type MVSSTART in an OS/2 window. This reads the necessary information from the MVSINFO.DAT file, starts SdU, and establishes the connection to the specified MVS system.

To see if the installations and configurations are done correctly, you can set up this command manually in an OS/2 window. Type `MVSSTART` in an OS/2 window. The first note you get does not belong to your system; ignore it and press Enter.

You are prompted to type in your password.

Next, the drive commands you specified in the MVSINFO.DAT file as drives to be attached are performed.

After successful attachment, SdU tests the connection to MVS beginning with trying to get time information from MVS. Then SdU tries to open a specific data set on MVS with the high-level qualifier SYS1, which exists in any case. When this works, you get a message about the successful connection and you can then access the attached data sets in the same way as you access drives on your workstation, on the LAN, or on an OS/2 window.

The following shows you the report you get while connecting to the host:

```
                  -------- Note --------
Depending on your system you may be prompted for your TSO
userID and/or password.  This prompting may occur for each
drive that is accessed.  You may initially get a message
that your password is invalid.  Ignore this message.

Press any key when ready . . .
DFM OS/2
Operating System/2
Starting Router
(C) Copyright IBM Corp. 1995. All rights reserved.
Version 2.0  Release 1.1  Level 00

Processing: RTYP CMGR
Processing: A2ET E:\IBMCOBOL\BIN\CONVTABL\04370500.CVT
Processing: E2AT E:\IBMCOBOL\BIN\CONVTABL\05000437.CVT
Processing: LCLN STB1835I

Processing: RMTN SC47APPC,COBRS08,The MVS/ESA System in POK
Enter password for system SC47APPC user ID COBRS08:

Processing: MODN QFSPC,QPCSUPP
Processing: RTDN SC47APPC

Default remote system name:  SC47APPC
6600 - Byte-stream Function already started.

Distributed FileManager (DFM/2) Version 1.00
STRTDFMC processing complete
Distributed FileManager (DFM/2) Version 1.00
STRTDFMR processing complete

Drive M: being accessed
File system = SDU
System = SC47APPC
High Level Qualifier = COBRS08
Usage = text (translation between EBCDIC and ASCII)
Name mapping = local using MVSINFO.DAT file
·E:\IBMCOBOL\MACROS‘cmd.exe /c dfmdrive assign M: COBRS08

Distributed FileManager (DFM/2) Version 1.00
EHN0230: Drive M: was successfully assigned
          to system SC47APPC,
          and directory: COBRS08.
EHN0232: Command completed successfully.
·E:\IBMCOBOL\MACROS‘cmd.exe /c dfmdrive setparm M: ″text,pc_ccsid(850)

Distributed FileManager (DFM/2) Version 1.00
EHN0251: Drive S: parameter list successfully changed
          to ,TEXT,PC_CCSID(850).
EHN0232: Command completed successfully.

Drive N: being accessed
File system = SDU
System = SC47APPC
```

```
High Level Qualifier = COBRS08
Usage = binary (no translation, do not use for editing)
Name mapping = local using MVSINFO.DAT file
•E:\IBMCOBOL\MACROS‘cmd.exe /c dfmdrive assign N: COBRS08

Distributed FileManager (DFM/2) Version 1.00
EHN0230: Drive N: was successfully assigned
          to system SC47APPC,
          and directory: COBRS08.
EHN0232: Command completed successfully.

Will test connection to MVS host: SC47APPC
UserID = COBRS08 , File system = SDU
Will attempt to get time information from MVS
 TIME-11:37:20 AM. CPU-00:00:00 SERVICE-1093 SESSION-00:00:00 MAY 5

Will attempt to get information for ′SYS1.MACLIB(OPEN)′
Data set = ′SYS1.MACLIB′
Member   = OPEN
Dsorg    = PO
Recfm    = FB
Lrecl    = 80
Alias    = NO

Connection to MVS host seems OK
Press any key when ready . . .

You should select ″View->Refresh now″ in order to update your
project view
Press any key when ready . . .
```

When the connection is established, an editor comes up in the background and reports that the access to the drive attached for edit is all right.

The connection to this drive is checked every 15 minutes by SdU, and this monitor reports it.

Time information from MVS

When your system tries to get time information from MVS and it fails, you can get more information about why it fails when you repeat this command in an OS/2 window:

```
mvsacall tso m: time
```

where m: is the attached drive you specified.

It is likely that SdU is not installed or configured properly on the host when you get a communication message.

Chapter 4. Creating Applications Using Remote E/C/D on OS/2

VisualAge for COBOL offers you a WorkFrame on the workstation for projects that reside on the host. You can set up the projects in the same way as you organize local projects on the PC.

To work with these host projects, you first configure the WorkFrame on the PC, then connect to the host with either TCP/IP or APPC. You can then work with the files in the projects as you are used to working with them in local projects.

4.1 Running a Sample Application on the Workstation

To show you how to handle projects in this way, we first provide an easy sample program that you can upload to the host, then edit, compile, run, and debug afterward.

4.1.1 Defining the Application on the Workstation

WorkFrame can recognize the extensions of the files residing on the host, submit the jobs, and show the system output of the submission. In that way, it is possible to compile and link all the projects.

For debugging a project, begin by starting a debug session on the workstation. This remains in wait status until it is called by the debugger on the host. A submitted job creates the connection between the debugger on the host and the session that was started. When this connection is established, you can debug the program directly from your workstation.

It is not necessary to log on before you create a WorkFrame project, but it is helpful to be logged on already. The logon on the host is actually only one command. When you have generated your WorkFrame project once, it becomes a selection in the menu bar. As long as you do not have a WorkFrame project for MVS, start the logon process from an OS/2 window by typing

```
MVSSTART
```

While the procedure is running, you have to press Enter several times. The NFSCLEAN procedure starts and tries to unmount drives. This fails as long as you have not logged on before using TCP/IP. Next, the NFSSTART runs and processes the logon to the host. The drives you specified in the MVSINFO.DAT file are mounted and, after successful mounts of all drives, the system tries to get time information from the host and then file information. If all these connections have been done successfully, you get the message that the connection seems to be OK. Also you get the message that you should click on **Refresh** to update the project view.

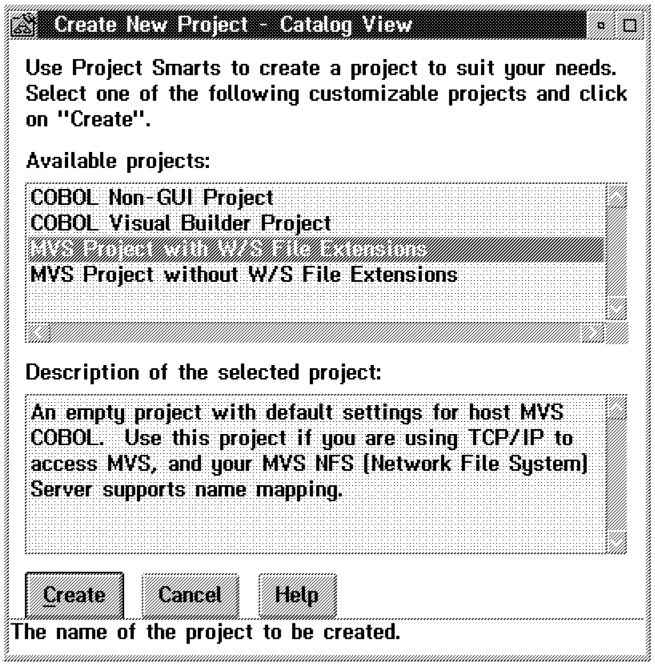
You do not have a project yet; that is our next step.

You can create a project for MVS in two different ways with the same result. You can either use the Create New Project icon to do it or use the Templates icon to do it. Both are located in the VisualAge for COBOL icon folder. We decided to describe using the Create New Project icon because it offers you more choices to set up your configuration for the WorkFrame. The steps are these:

1. To create an MVS project, double-click on the VisualAge for COBOL icon located on the desktop. The upcoming window shows a **Create New Project** icon. Double-click on it. The **Create New Project - Catalog View** window (Figure 50 on page 60) appears.

   If your MVS NFS supports file name mapping, select **MVS Project with W/S File Extensions** in the *Available projects* list (Figure 50). As long as you are not sure whether the MVS NFS supports the file name mapping or not, select **MVS Project without W/S File Extensions**, but make sure that the host gets this feature as soon as possible because it is important for you.

   If you are using APPC for the connection, select **MVS Project without W/S File Extensions**.



Use Project Smarts to create a project to suit your needs. Select one of the following customizable projects and click on "Create".

Available projects:

- COBOL Non-GUI Project
- COBOL Visual Builder Project
- MVS Project with W/S File Extensions
- MVS Project without W/S File Extensions

Description of the selected project:

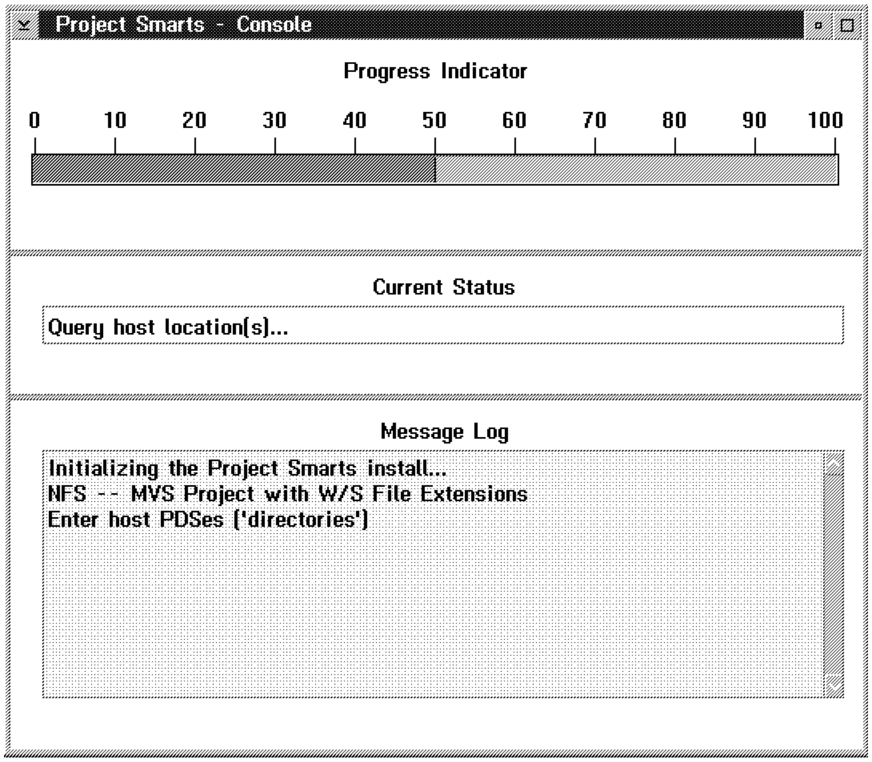
An empty project with default settings for host MVS COBOL. Use this project if you are using TCP/IP to access MVS, and your MVS NFS (Network File System) Server supports name mapping.

The name of the project to be created.

Figure 50. Create New Project - Catalog View Window

   After selecting this project, click on **Create**.

2. The **Project Smarts - Console** window comes up and shows you the status of the generation of the project (Figure 51 on page 61).

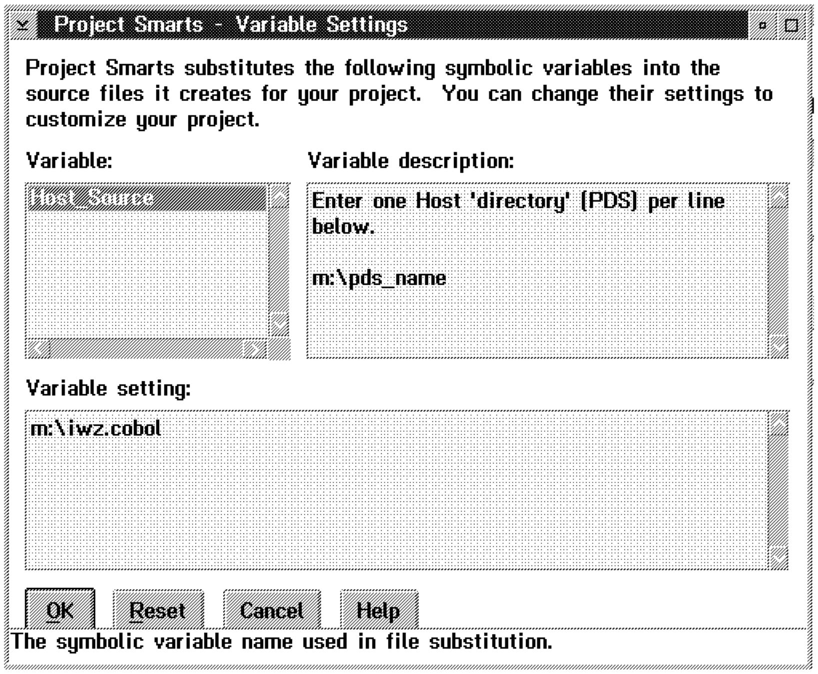


Query host location(s)...

Initializing the Project Smarts install...
NFS -- MVS Project with W/S File Extensions
Enter host PDSes ('directories')

Figure 51. Project Smarts - Console Window

Next, the **Project Smarts - Variable Settings** window appears (Figure 52).



Project Smarts substitutes the following symbolic variables into the source files it creates for your project. You can change their settings to customize your project.

Variable: Host_Source

Variable description: Enter one Host 'directory' (PDS) per line below.

m:\pds_name

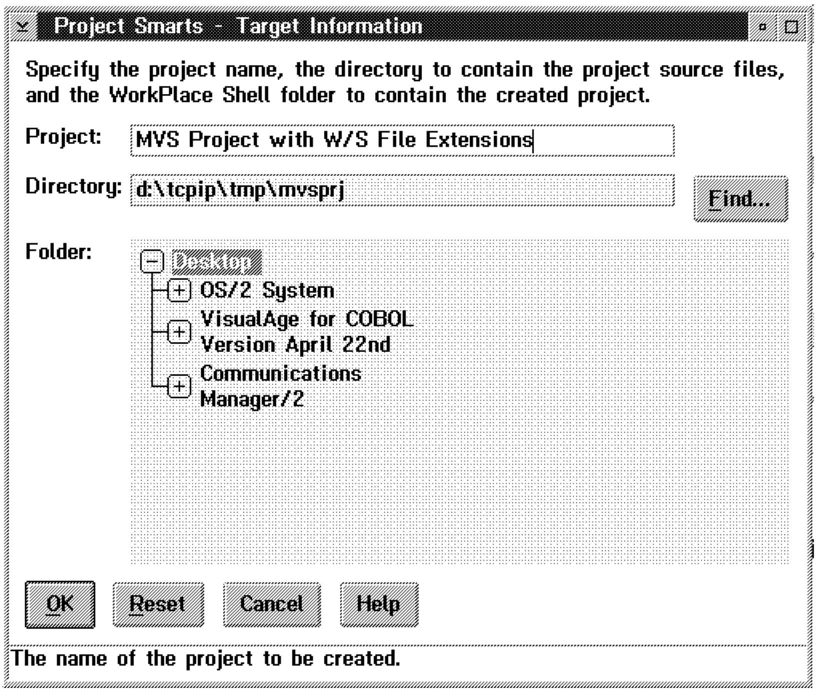
Variable setting: m:\iwz.cobol

The symbolic variable name used in file substitution.

Figure 52. Project Smarts - Variable Settings Window

You can leave the settings in the window as is and click on **OK**.

3. The **Project Smarts - Target Information** window comes up (Figure 53 on page 62).



Project Smarts - Target Information

Specify the project name, the directory to contain the project source files, and the WorkPlace Shell folder to contain the created project.

Project: MVS Project with W/S File Extensions

Directory: d:\tcpip\tmp\mvsprj

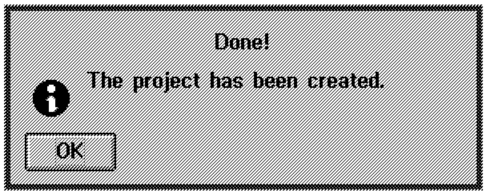
Folder:
- Desktop
  - OS/2 System
  - VisualAge for COBOL Version April 22nd
  - Communications Manager/2

The name of the project to be created.

Figure 53. Project Smarts - Target Information Window

Here you can specify the project name, the directory on the workstation for the temporary files and the folder in which the project should reside. Remove the entry from the *Project* entry field and insert the name you want to give your project. We called it **MVS Project Hello**. You can leave the directory for the temporary files as is, or you can change it to have your own organization of files and projects. Then you can also select the folder where the project icon should be located. The default is the desktop. After making these configurations, click on **OK**.

4. The **Done!** window tells you that the project has been created (Figure 54).



Done!

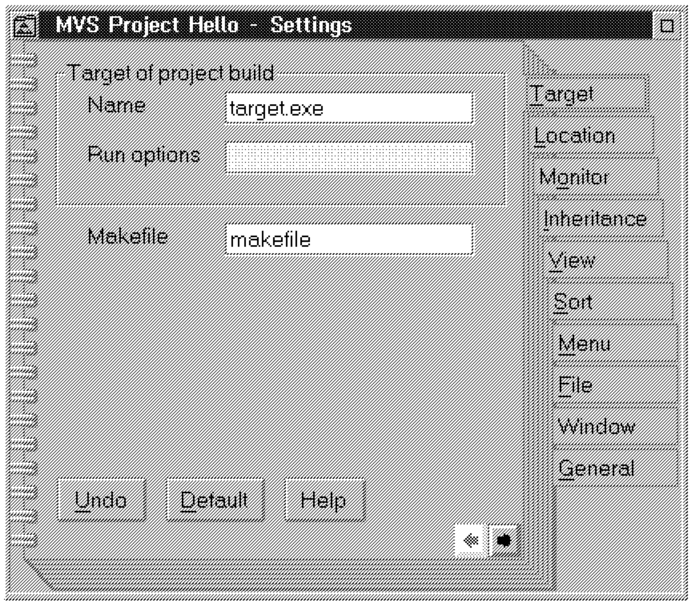
The project has been created.

Figure 54. Done! Window

Click on **OK** to close this message window. You have now created your first project to work remotely on the host.

5. Close the remaining windows that you no longer need. To close the **Create New Project - Catalog View** window click on **Cancel** and also close the VisualAge for COBOL icon view window.

6. To set up the settings of your project, move your mouse pointer to the COBOL MVS Local Project icon on the desktop you just created and click mouse button 2. A pop-up menu appears. Select **Settings** from the pop-up menu. The **MVS Project Hello - Settings** window is displayed (Figure 55).


MVS Project Hello - Settings

Target of project build

Name: target.exe

Run options:

Makefile: makefile

Figure 55. MVS Project Hello - Settings Window

Scroll to the next page and leave the Target page as is, because this page is valid for local projects only when you create a local execution file.

7. On the *Location* page you see the *Source directories for project files* list. In this list are two paths already qualified: a local path for temporary file savings on the workstation, and the default path where the host source code could reside. Here is the drive that was specified in the *Variable Setting* field on the **Project Smarts - Variable Setting** window, if you did not customize the variable settings in this window. The default drive and path for this is `m:\iwz.cobol`. Change this path into the path where the COBOL code resides on the host.

   Also, specify all data sets you want to see in the WorkFrame—for example, the JCL files and copy books.

   The files that should be included in the MVS project are at least of two different types. You probably want to see

   - Source COBOL files (to edit them) using the editor of the WorkFrame
   - JCL files, which can be submitted directly from the WorkFrame to compile, to link, and to debug the source COBOL files.

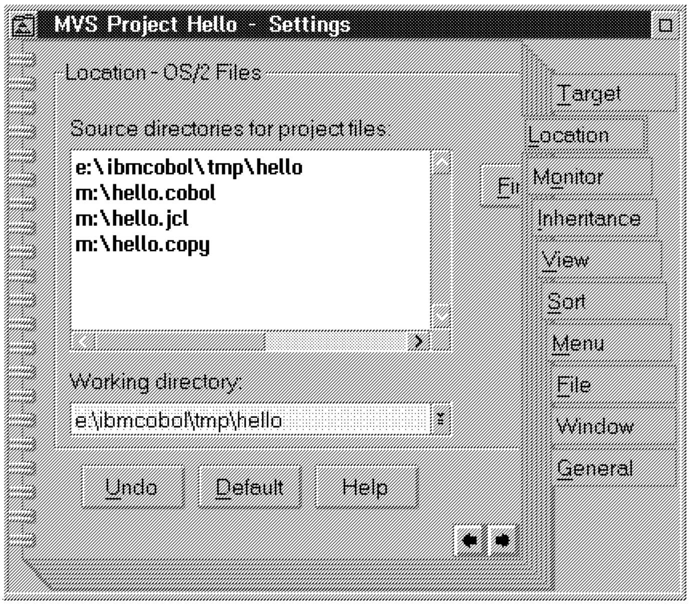
   In our case, the COBOL files reside on MVS in the data set called *COBRS08.HELLO.COBOL*, where the high-level qualifier is our user ID, COBRS08. We specified this high-level qualifier before in the MVSINFO.DAT file as our drive letter m:.

   We thus declare the data sets for the COBOL source code, the JCLs, and the copy books as shown in Figure 56 on page 64. If you follow our example, type these four lines into the Source directories for project files list:

```
e:\ibmcobol\tmp\hello
m:\hello.cobol
m:\hello.jcl
m:\hello.copy
```

With that, we also change the temporary directory to ibmcobol\tmp\cobol. Also, change the entry in the *Working directory* field to this temporary directory.

Do not define any data set you access in binary mode with the drive letter n: For these files, you do not get a file extension in the WorkFrame project.



```
e:\ibmcobol\tmp\hello
m:\hello.cobol
m:\hello.jcl
m:\hello.copy
```

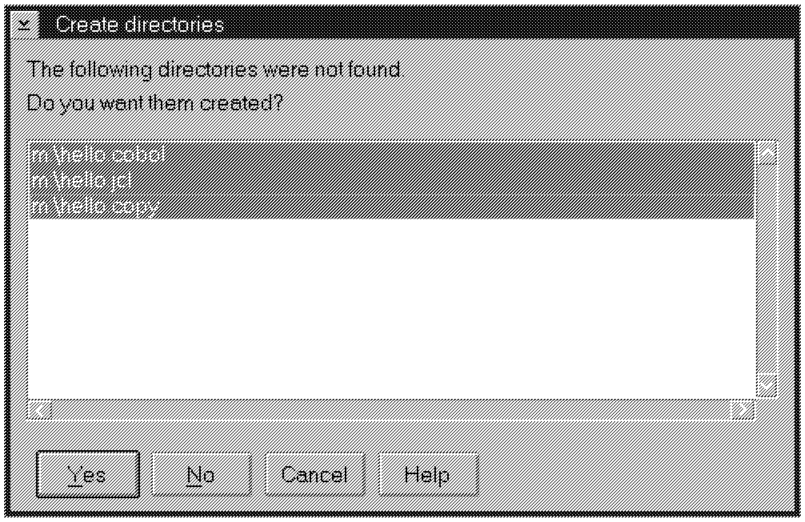
e:\ibmcobol\tmp\hello

Figure 56. MVS Project Hello - Settings Window: Location Page

Note

All members of the data sets you specify here appear in the WorkFrame after you log on the host. You cannot enter any data set whose high-level qualifier is not specified in the MVSINFO.DAT file you configured before.

When you specify a drive and a path that does not exist and you press Enter, you are prompted by the **Create directories** window (Figure 57 on page 65).


Create directories

The following directories were not found.

Do you want them created?

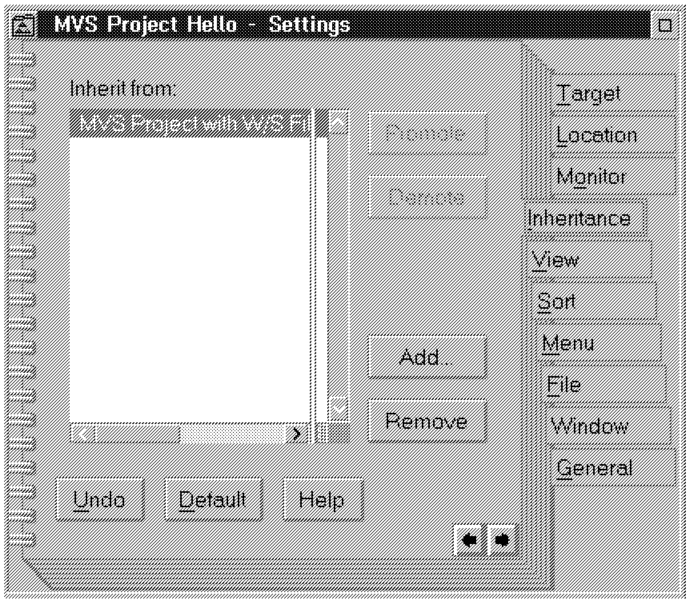
m:\hello.cobol
m:\hello.jcl
m:\hello.copy

Figure 57. Create directories Window

Do not let these directories be created by the tool and click on **No** or **Cancel**. This window is being closed. You want to create the directories yourself, give your own the data set specifications, and not take the defaults.

Further ensure that the *Working directory* field on the Location tab specifies a local directory.

8. Scroll to the *Inheritance* tab and ensure that the inheritance from MVS Project with W/S File Extensions Master Project is (Figure 58) for a project to connect via TCP/IP. A project with APPC connection to the host has to inherit from MVS Project without W/S File Extensions Master Project.


MVS Project Hello - Settings

Inherit from:

MVS Project with W/S Fil

Figure 58. MVS Project Hello - Settings Window: Inheritance Page

If this is not the case, select the project type that is given there and click on **Remove**. The default settings for the project are then removed. Click on

Add... and the window **Select a Project to Inherit From** appears. If your hard disk drive (the boot drive for OS/2) is File Allocation Table (FAT) formatted, select **D:\DESKTOP\VISUALAG\WORKS\IWZVMVSN** for a project with file-name mapping or **D:\DESKTOP\VISUALAG\WORKS\IWZVMVSL** for a project without file-name mapping where D is your OS/2 boot drive and has the file your project inherits from. If your hard disk drive D is high-performance file system (HPFS) formatted, select **D:\Desktop\VisualAge COBOL\Works\MVS Project with W/S File Extensions Master Project** or **D:\Desktop\VisualAge COBOL\Works\MVS Project without W/S File Extensions Master Project** depending on your host. Click on **Inherit**.

Close the **MVS Project - Settings** window. Your Hello project should now be set correctly.

You have created a WorkFrame with the settings of the project, but you still need the host files you want to access and execute.

9. To copy the necessary files from the diskette to the host, follow these instructions:

   - First, create the data sets on the host because they probably do not exist in your environment. To allocate them, open an OS/2 window and switch to drive letter m: by typing

     ```
     m:
     ```

     and press Enter. You can access the host drive with your user ID as the high-level qualifier in the same way as you access the local drives of your workstation.

   - To create the three directories, type the following commands in sequence in this OS/2 window, and press Enter:

     ```
     md "hello.cobol,dsntype(pds),lrecl(80),blksize(6160),recfm(fb)"
     md "hello.jcl,dsntype(pds),lrecl(80),blksize(6160),recfm(fb)"
     md "hello.copy,dsntype(pds),lrecl(80),blksize(6160),recfm(fb)"
     ```

     You must give some data set attributes as parameters to allocate the right kinds of data sets. Or, you can use the emulation to allocate the new data sets with these specifications:

Space units	BLOCK
Primary quantity	112
Secondary quantity	10
Directory blocks	30
Record format	FB
Record length	80
Block size	6160

Data Set Creation Attributes

To see a list of all the parameters you can use to specify data-set attributes, type the command

```
showattr hostname |more
```

into an OS/2 window. The first part of the information you get is what is of interest:

```
DFSMS/MVS 1.3.0 Network File System Server Data Set
Creation Attributes:

lrecl(8196)           recfm(vb)             blksize(0)
space(100,10)         blks                  dsorg(ps)
dir(30)               unit(sysallda)        volume()
recordsize(512,4K)    keys(64,0)            nonspanned
shareoptions(1,3)
mgmtclas()            dsntype(pds)          norlse
dataclas()            storclas()
```

- After creating the data sets, you can copy the files from the diskette to the host or upload them with the emulation. To copy them from the diskette, type the commands in the OS/2 window as follows:

  ```
  copy a:\hello\cobol\dttest.cbl  m:\hello.cobol\dttest
  copy a:\hello\cobol\hello.cbl   m:\hello.cobol\hello
  copy a:\hello\jcl\dtbld.jcl     m:\hello.jcl\dtbld
  copy a:\hello\jcl\dtrun.jcl     m:\hello.cobol\dtrun
  copy a:\hello\copy\cblatc8.cpy  m:\hello.copy\cblatc8
  ```

  Unfortunately you cannot copy the files from the diskette to the host by using a wild card character to copy more than one file using one command. If you try this, you get the OS/2 error message “SYS0027: The drive cannot find the sector (area) requested.” This message text does not make sense, but it tells you that you cannot access a data set in this way.

  If you have TCP/IP installed, you also can copy the files into the data sets using FTP as described in 2.4, “Upload Workstation Files Using FTP” on page 25.

10. Open the new project by double-clicking on its icon on the desktop. The WorkFrame opens but there are still no files in it. Select **View** in the menu bar of the WorkFrame and then click on **Refresh now** to display the data, or press **F5**. The WorkFrame project should show the five files as shown in Figure 59 on page 68.

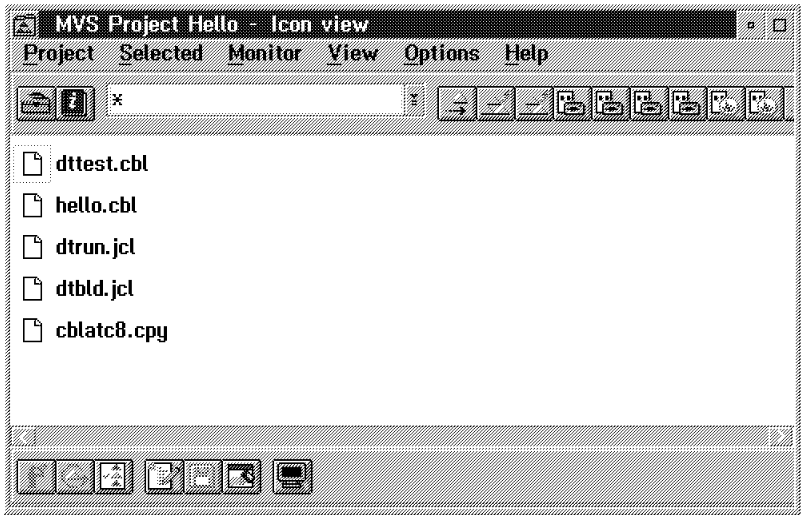

MVS Project Hello - Icon view
Project Selected Monitor View Options Help
*
dttest.cbl
hello.cbl
dtrun.jcl
dtbld.jcl
cblatc8.cpy

Figure 59. MVS Project Hello - Icon view Window

Log on from the WorkFrame project

If you are not logged on, select **Project** in the menu bar, select the arrow of **MVS Access**, and then click on **Connect MVS drives** to connect to the host. After a successful connection, press **F5** and the files belonging to the specified data sets will appear in the WorkFrame.

If some files do not have a file extension even though the file extension mapping is supported from your TCP/IP and NFS, you should check whether the low-level qualifier of these members is resolved in the NFS mapping file on the host. If not, add a line for it in your NFS.MAPPING file. To make these changes valid in your WorkFrame, unmount the drive and mount it again using the **Disconnect MVS drives** function supplied by the WorkFrame.

After each change of the drive specification in the project settings, press F5 to refresh the WorkFrame.

Drive specifications

As mentioned before, you still have the opportunity to change the drive settings you made in the MVSINFO.DAT file. To do so, click on **Project** in the menu bar and select **MVS Setup** in the pull-down menu. The editor starts and displays the MVSINFO.DAT file. Here you can change or add drives in the same way you specified them before. After that, save and close the file.

After a change of the drive specification in the MVSINFO.DAT file, you must log off the host and log on again to activate the changes and make available the new data sets you set up. To log off, use the function provided by the WorkFrame. You reach this function by selecting **Project** in the menu-bar, selecting the arrow of **MVS Access**, and then clicking on **Disconnect MVS drives**.

Now that you have created and set up your project, you can edit the files on the workstation, compile the programs by submitting jobs, and debug the COBOL programs on the workstation, as described in the subsequent sections.

4.1.2 Editing, Compiling, and Running the Application Remotely

You can now edit all the files you see on the WorkFrame after you log on.

To work with a file in the WorkFrame, select it and click on **Selected** in the menu bar. The pull-down menu shows you a list of actions belonging to the selected file. If you selected, for example, a source file, a JCL, or a copy file, you can edit it. When you select a file with mouse button 2 and you select **MVS Edit** in the pop-up menu, the MVS editor comes up and you can edit and save the file you are working with.

To practice, select the file **hello.cbl** and click mouse button 2 or **Selected** in the menu bar. Selecting mouse button 2 gives you more choices than Selected, because you also get the standard functions for files like copying or deleting the file (Figure 60).

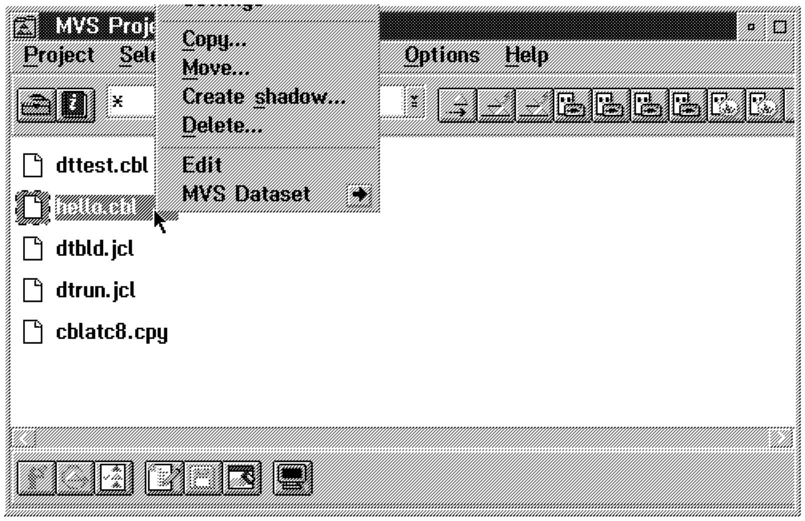


Figure 60. MVS Project Hello - Icon View Window with Pop-up Menu

Select **Edit** in the pop-up menu. The resulting window reports

```
Accessing MVS data set 'COBRS08.HELLO.COBOL(HELLO)'
Accessing MVS to get data set/member information
```

Normally, the MVS editor appears, showing the selected file.

Important

It could happen that you get the error message:

```
Cannot determine data set information for M:\HELLO.COBOL\hello.cbl
Press any key when ready . . .
```

This tells you that you allocated a data set with the wrong data set specifications. When you allocate, for example, a data set with variable block size, you cannot access the files using the MVS editor.

The MVS editor is active and you can use it in the same way as you operate with the LXPM editor or any other editor you know.

You can also change the key behavior of the editor so that you can use it like the editor of ISPF (for example). To do that, select **Options** in the menu bar, then select the arrow of **Key behavior** in the pull-down menu, and then select ISPF. Line numbers are added on the left side of the window in the text and you can use them for the line commands you know from ISPF on the mainframe. Change the key behavior of the editor as you like.

To make some changes in the code, we recommend following our example by adding a display statement and a calculation, as we did in Figure 61.

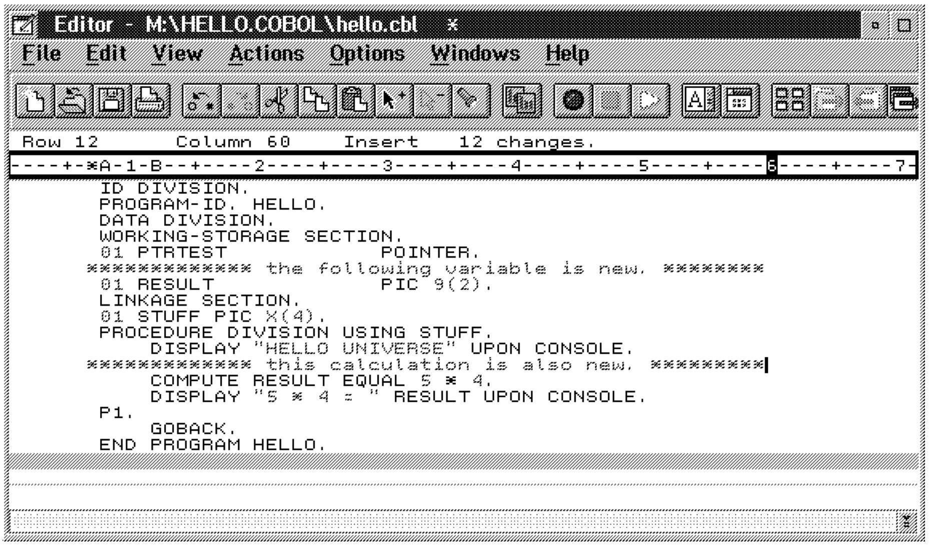
Add some commands as described below:

1. Go to the Working-Storage section and add the variable

```
01 RESULT                    PIC 9(2).
```

2. Go to the Procedure Division and add the statements under the Display statement

```
COMPUTE RESULT EQUAL 5 * 4.
DISPLAY "5 * 4 = " RESULT UPON CONSOLE.
```


Editor - M:\HELLO.COBOL\hello.cbl
File Edit View Actions Options Windows Help

```
Row 12       Column 60     Insert    12 changes.
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-
       ID DIVISION.
       PROGRAM-ID. HELLO.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 PTRTEST             POINTER.
      ************* the following variable is new. ********
       01 RESULT              PIC 9(2).
       LINKAGE SECTION.
       01 STUFF PIC X(4).
       PROCEDURE DIVISION USING STUFF.
           DISPLAY "HELLO UNIVERSE" UPON CONSOLE.
      ************* this calculation is also new. *********
           COMPUTE RESULT EQUAL 5 * 4.
           DISPLAY "5 * 4 = " RESULT UPON CONSOLE.
       P1.
           GOBACK.
       END PROGRAM HELLO.
```

Figure 61. Editor - M:\HELLO\COBOL\hello.cbl Window

After making these changes, select **File** in the menu bar and **Save** in the pull-down menu. On the bottom of the editor window, a message is displayed: “Performing a protected save, please wait.” This save takes some seconds because of the special way of saving the file.

When you get the message “Protected save completed OK,” you can close the editor by double-clicking on the icon in the upper left corner of the window. While closing the window, the temporary file from this data set member gets deleted because it is no longer needed.

To work with the editor of the WorkFrame without using WorkFrame, you can type MVSEDIT in an OS/2 window. You get the same editor.

Editing MVS files restriction

You cannot edit MVS files that contain characters whose hexadecimal value is below X′40′. These represent unprintable control characters that cannot be converted between EBCDIC and ASCII for editing at the workstation. These also include files that contain the DBCS shift-out (X′0E′) and shift-in (X′0F′) characters.

When you try to edit such a file, the editor opens and displays the contents of the file, but remember that the editor opens a temporarily created copy of the original file. Also you get the following message at the beginning of the file:

```
INFO -------------------- Start of MVS data set information -------
INFO
INFO The editing session has been set to read only to prevent
INFO the data set from being modified due to the following:
INFO
INFO - The data set contains characters whose hex positions
INFO   are below hex 40.  These characters cannot be
INFO   converted between MVS EBCDIC and workstation ASCII
INFO   for editing at the workstation.  Note that the file
INFO   you are viewing is a copy of your original file where
INFO   these characters appear as û
INFO
INFO --------------------- End of MVS data set information --------
```

However, you can browse this file with the editor.

The restriction is set by the MVS editor. When you open this file from an OS/2 window, using the normal OS/2 editor, you can edit and save it. But you have to be aware that you change these hexadecimal values when you save the file and that this editor does not provide protected saving. When the file is getting too big to be saved in the same data set, an incomplete file is saved.

To run the application on the host, you have to submit the job in the same way as you did it on the host. So if you select a JCL file on WorkFrame, you get the choice to submit this job.

Now open the file dtbld.jcl to edit it. You see a JCL that is ready to compile and link the programs together. You see some error messages in the first lines of the JCL. Either you change the language profile or you ignore the messages. To change the language profile, click on **Options** in the menu bar and on **Profiles** in the pull-down menu and then select **Change profile**. The **Change profile** window comes up (Figure 62).

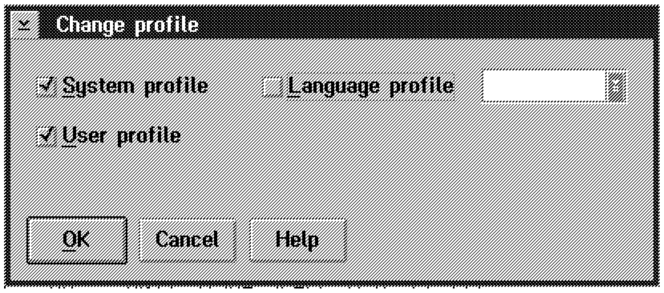


Figure 62. Change profile Window

Disable the check-box *Language profile* and click on **OK**. The code changes its color to black, the editor no longer reads your code, and the error messages disappear.

The program DTTEST calls HELLO, so both programs have to be compiled and linked. You have to change the job card at the beginning of the JCL that it fits to your host environment and the user ID referenced by the variable &USRPRFX.

Also, before you run the JCL, ensure that you have allocated all the data sets we are using in this JCL. Either you allocate them by typing the following commands in an OS/2 window

```
md ″hello.listing,dsntype(pds),lrecl(133),blksize(133),recfm(fba)″
md ″hello.sysadata,dsntype(pds),lrecl(1020),blksize(1024),recfm(vb)″
md ″hello.load,dsntype(pds),lrecl(255),blksize(6160),recfm(u)″
```

or you change the JCL to fit your system.

If you want to allocate the data sets using an emulation session, allocate them with these specifications:

- Allocate the HELLO.LISTING data set with these attributes at least of this size:

Space units	BLOCK
Primary quantity	2232
Secondary quantity	128
Directory blocks	7
Record format	FBA
Record length	133
Block size	133

- Allocate the HELLO.SYSADATA data set with these attributes, at least of this size:

Space units	BLOCK
Primary quantity	132
Secondary quantity	10
Directory blocks	0
Record format	VB
Record length	1020
Block size	1024

- Allocate the HELLO.LOAD data set with these attributes, at least of this size:

Space units	BLOCK
Primary quantity	112
Secondary quantity	10
Directory blocks	30
Record format	U
Record length	255
Block size	6160

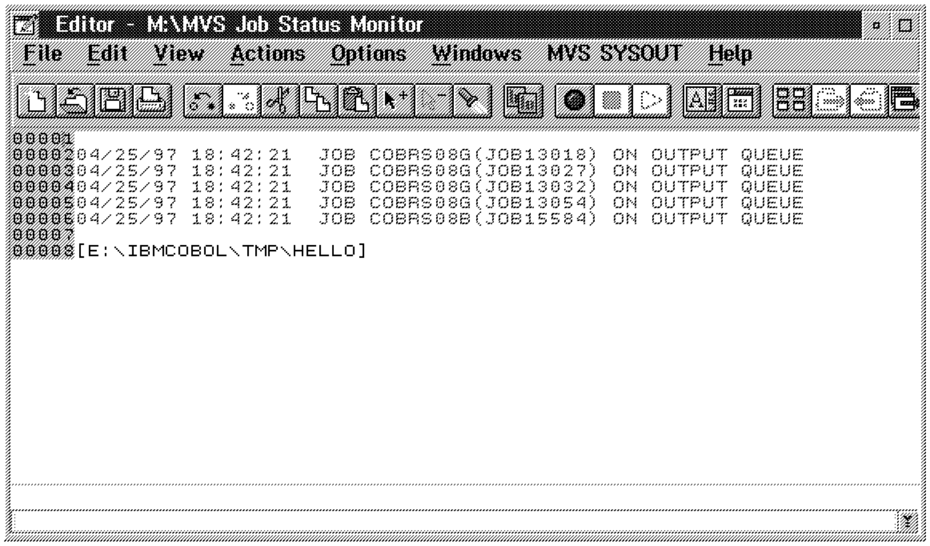
Also, check if your COBOL compiler is located in the data set IGY.V2R1M0.SIGYCOMP. If not, change the data set name in the JCL.

To submit the job, select file **dtbld.jcl** and click on mouse button 2. Then select **Submit job on MVS** in the pop-up menu. The WorkFrame is then divided into two parts. The upper part shows you the file list as you have seen it before, and the lower part changes to a message window that keeps you informed about the actions of the running job. You get the following messages about this submission:

```
Starting Submit Job on MVS::Submit job on MVS
mvssub.cmd M:\HELLO.JCL\dtbld.jcl
Submitting ′COBRSO8.HELLO.JCL(DTBLD)′ on WTSC47
===> JOB COBRSO8B(JOB15584) SUBMITTED

Action complete, RC=0
```

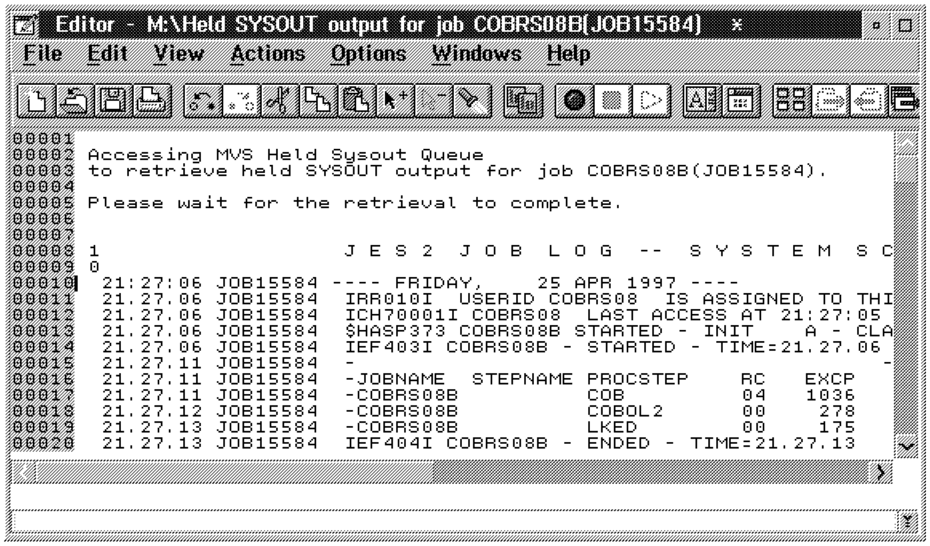
After this submission, you want to see the status of the submitted job. The ″Action complete, RC=0″ message that appears in the monitor of your WorkFrame tells you only that the job got submitted correctly. It does not say anything about the result of the submission. For that, select **Project** in the menu bar, then select the arrow of **MVS Tools** in the pull-down menu, and select **MVS job status**. The **MVS Job Status Monitor** window comes up in the same way as you are used to seeing it in MVS (Figure 63).



```
00001
0000204/25/97  18:42:21  JOB COBRS08G(JOB13018) ON OUTPUT QUEUE
0000304/25/97  18:42:21  JOB COBRS08G(JOB13027) ON OUTPUT QUEUE
0000404/25/97  18:42:21  JOB COBRS08G(JOB13032) ON OUTPUT QUEUE
0000504/25/97  18:42:21  JOB COBRS08G(JOB13054) ON OUTPUT QUEUE
0000604/25/97  18:42:21  JOB COBRS08B(JOB15584) ON OUTPUT QUEUE
00007
00008[E:\IBMCOBOL\TMP\HELLO]
```

Figure 63. MVS Job Status Monitor Window

Make sure that your last job is already displayed in the list. If not, select **MVS SYSOUT** in the menu bar and then **Refresh**. Select the job you want to see, and then **MVS SYSOUT** and **List contents** to read the job listing (Figure 64 on page 74).


Editor - M:\Held SYSOUT output for job COBRS08B(JOB15584)
File Edit View Actions Options Windows Help

```
Accessing MVS Held Sysout Queue
to retrieve held SYSOUT output for job COBRS08B(JOB15584).

Please wait for the retrieval to complete.


1                   J E S 2   J O B   L O G   --   S Y S T E M   S C
0
 21:27:06 JOB15584 ---- FRIDAY,     25 APR 1997 ----
 21.27.06 JOB15584  IRR010I  USERID COBRS08  IS ASSIGNED TO THI
 21.27.06 JOB15584  ICH70001I COBRS08  LAST ACCESS AT 21:27:05
 21.27.06 JOB15584  $HASP373 COBRS08B STARTED - INIT    A - CLA
 21.27.06 JOB15584  IEF403I COBRS08B - STARTED - TIME=21.27.06
 21.27.11 JOB15584  -                                          -
 21.27.11 JOB15584  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP
 21.27.11 JOB15584  -COBRS08B          COB         04   1036
 21.27.12 JOB15584  -COBRS08B          COBOL2      00    278
 21.27.13 JOB15584  -COBRS08B          LKED        00    175
 21.27.13 JOB15584  IEF404I COBRS08B - ENDED - TIME=21.27.13
```

Figure 64. Editor - M:\Held SYSOUT output for job Window

Note

The job can be too big to display in this window when the compile fails and the dump becomes too long. Then the editor window opens with an error message saying OUTPUT ENDED DUE TO ERROR, SYSTEM ABEND CODE B37 Remember that this error message does not mean the abend of the job.

When the program is compiled correctly, you can submit the JCL **dtrun.jcl** to start the program in the same way as you submitted the compile job. This sample HELLO is only a batch program and you have to view the system output to get the displayed messages.

The MVS Job Status monitor is provided for monitoring host batch jobs that you have submitted. The monitor displays the status of jobs whose job name follows the time sharing option (TSO) convention. Each line of the job status is prefixed with the date and time of day. The monitor uses the TSO STATUS command (with no parameters) to obtain the job status information.

Start the MVS Job Status Monitor by selecting **Project**, then **MVS Tools**, then **MVS Job Status** from your project′s menu bar. For those jobs that are on a held SYSOUT output queue (the job status contains the string ON OUTPUT QUEUE), you can perform the following actions by first pointing the cursor to the line with the job and then selecting the **MVS SYSOUT** menu bar choice:

- **List contents**

  Lists the entire contents of the held SYSOUT in an Editor window and includes any carriage control that is present in column 1. For display purposes, any hexadecimal characters that are below X′40′ are translated to blanks. The monitor uses the TSO OUTPUT command to obtain the SYSOUT output contents. The SYSOUT output remains in the output queue after being listed.

- **List 100 lines**

The action is similar to that of list contents, except only the first 100 lines of the held SYSOUT are listed. If your host system has customized exits that include the status of each step at the beginning of the SYSOUT, listing the first 100 lines allows you to see if the job ran successfully without the overhead imposed by listing the entire contents.

- **Save as jobname**

  The held SYSOUT output is saved to the partitioned data set with the fully qualified name of userid.IWZ.OUTLIST(member) where the member name is the job name. The TSO OUTPUT command is used to save the output. The SYSOUT output is deleted from the output queue after being saved. After the save, the job status information is automatically refreshed. To ensure that the userid.IWZ.OUTLIST data set has sufficient space to hold the output of many jobs, allocate the data set before using the MVS Job Status monitor. If the data set is allocated as a partitioned data set (PDS) (as opposed to a partitioned data set extended (PDSE)) you will need to compress the data set occasionally.

- **Save as jobnum**

  The action is similar to that of the Save as jobname, except that the member name is the job number instead of the job name. After the save, the job status information is automatically refreshed.

- **Delete**

  The SYSOUT is deleted from the output queue without being saved or printed. You should list or save the SYSOUT contents before using this action. The TSO OUTPUT command is used to delete the output. After the delete, the job status information is automatically refreshed.

- **Refresh**

  Causes the job status information to be refreshed with the latest job information from the host.

To terminate the MVS Job Status monitor, close the window in which the monitor is running. If you get an Editor message requesting you to stop an active program, reply Yes.

Compress the data set

When you get the error message

```
     SYSTEM COMPLETION CODE=E37  REASON CODE=00000004
```

after a job submission, it means that the data set containing the listings is full and must be compressed. You can compress any data sets whose members the WorkFrame displays. To compress the data set COBRS08.HELLO.LISTING, add it in the settings for the WorkFrame on the location page. To do so, click on the icon in the upper left corner of the WorkFrame and select **Settings** in the pull-down menu. Scroll to the **Location** tab in the settings notebook and add **m:\hello.listing** in the list *Source directories for project files.* Close the settings and the members of the specified data set appear in the Workframe.

Now click on **dttest.lst** with mouse button 2, select the arrow of **MVS Dataset** in the pop-up menu, and click on **Compress PDS**. A window then comes up with the following messages:

```
     Will send to execute on WTSC47: CMPR 'COBRS08.HELLO.LISTING(DTTEST)
     Will execute on TSO: IGYFCMPR 'COBRS08.HELLO.LISTING(DTTEST)'

     User data set 'COBRS08.HELLO.LISTING' has been backed up to backup
     data set 'COBRS08.IWZ.@@TEMP@@.BACKUP'

     The backup will be deleted if the compress is successful.

     User data set 'COBRS08.HELLO.LISTING' compressed successfully
     TSO return code = 0

     Press any key when ready . . .
```

Now your data set has free space and you can submit the JCL again.

When you have compiled your program successfully, if it does not work as designed, you can debug the application. This is described next.

4.1.3 Debugging the Application

This section describes how you can debug a COBOL program that resides on the host using the debug function for host programs supported by VisualAge for COBOL.

The debug tool is one of the most used functions because of its extensive functionality and because it is easier than the common host debug tools.

To debug a program, you must compile it with a special parameter. Only if it was compiled with this parameter can the debugger catch the program and display its listing. Without having this option set, the program would be executed without being stopped by the debugger and you could not step through it.

The job that compiles and links the programs has to have the parameter 'TEST' in the parameter list of the compile step. An example of this compile command with the parameter list is as follows:

```
     COBOL2   EXEC  PGM=IGYCRCTL,REGION=4096K,
     PARM='APOST,LIB,OBJECT,RENT,TEST,ADATA'
```

Also, the JCL that must be submitted to run the application needs a special parameter. For TCP/IP, this is the IP address of the workstation on which you submit the job. The REGION command has to be extended with the parameter

```
PARM=('/TEST(,,,VADTCPIP&YOUR.IP.ADDRESS:*)')
```

where YOUR.IP.ADDRESS is the IP address of your PC, such as 9.112.34.49.

For APPC, the parameter needed is the APPC symbolic destination name of the workstation on which you submit the job. The REGION command has to be extended with the parameter

```
PARM=('/TEST(,,,VADAPPC&MACHINENAME:*)')
```

where MACHINENAME is the symbolic destination name of your PC, such as STB1835.

The JCLs belonging to the sample HELLO are already supplied with this parameter, but you have to change the IP address to yours, and change to your user ID.

You can find the IP address of your workstation in the TCP/IP configuration. To reach this, open the OS/2 System folder by double-clicking on the appropriate icon on the desktop. Then double-click on the **TCP/IP** icon in this folder and finally on the **TCP/IP Configuration** icon that comes up in the folder. The **TCP/IP Configuration** notebook appears. The first page that has the Network tab contains the variable for the IP address recorded.

Besides the additional parameter for the IP address, one of the STEPLIB DD statements in this JCL has to specify the data set for the debugger. You should ask your system administrator for the name of this data set. In our configuration, the debugger data set resides in the data set named EQAW.V1R2M0.SEQAMOD.

A sample for this part of the JCL is for TCP/IP:

```
// REGION=32000K,PARM=('/TEST(,,,VADTCPIP&9.112.34.49:*)')
//STEPLIB     DD  DSNAME=CEEV1R50.SCEERUN,DISP=SHR
//            DD  DSNAME=COBRS08.CBL.LOAD,DISP=SHR
//            DD  DSNAME=EQAW.V1R2M0.SEQAMOD,DISP=SHR
```

For APPC the REGION command differs as follows:

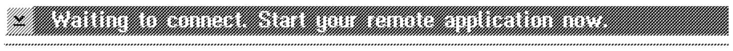
```
// REGION=32000K,PARM=('/TEST(,,,VADAPPC&STD1835:*)')
```

After preparing the JCL for the use of the debugger tool and submitting the JCL that builds the load module, you can run the debugger.

Follow the instructions for debugging the HELLO sample program to get some practice with the debugger tool.

1. Select the **dtbld.jcl** icon in the WorkFrame and click mouse button 2. Select **Submit Job on MVS** to build the application.
2. After a successful build, start the debugger by first selecting **Project** in the menu bar, then select the arrow of **Start Debug Session** and select the communication protocol you are using.

   The **Waiting to connect** window appears in the middle of your monitor (Figure 65 on page 78).



Waiting to connect. Start your remote application now.

Figure 65. Waiting to connect Window

3. Now submit the job that runs the application. In the HELLO sample, it is the JCL dtrun.jcl. To do so, select the JCL job and click on mouse button 2. On the pop-up menu, select **Submit Job on MVS**.

   The **Debugger - Session Control** window comes up first (Figure 66).

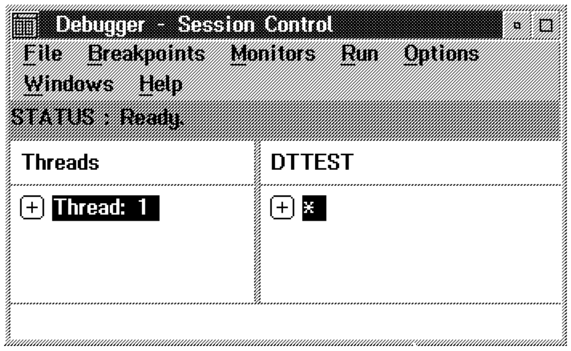
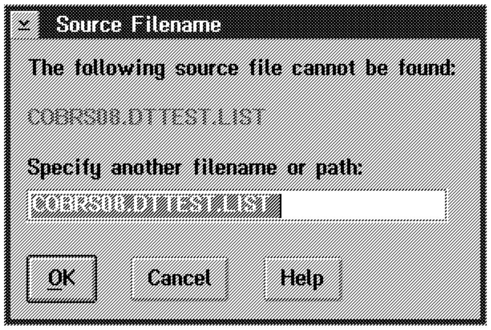

Debugger - Session Control
File Breakpoints Monitors Run Options
Windows Help
STATUS : Ready.
Threads
DTTEST
Thread: 1
*

Figure 66. Debugger - Session Control Window

It can happen that the debugger cannot find the source of the program to be displayed and debugged. Then a window appears and requests you to specify the path and the file name of the listing of the program (Figure 67).



Source Filename

The following source file cannot be found:

COBRS08.DTTEST.LIST

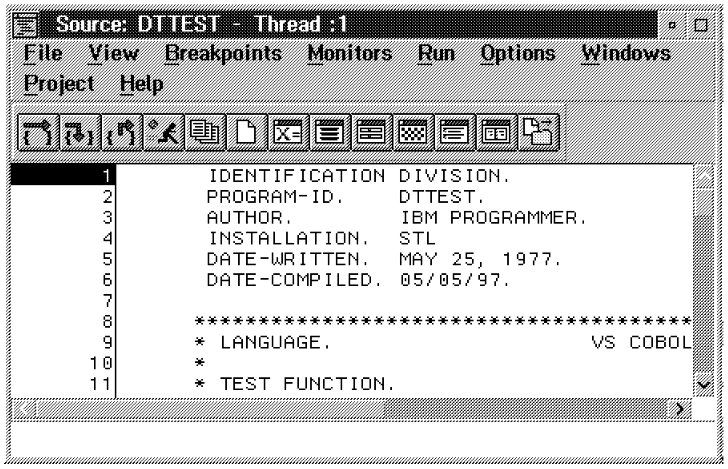
Specify another filename or path:

COBRS08.DTTEST.LIST

Figure 67. Source Filename Window

If this occurs, your JCL that built the application is incomplete. Compare it with the JCL DTBLD.JCL on the diskette that belongs to this book. You find it in the path A:\HELLO\JCL. To continue with the debugging, type in the path `COBRS08.HELLO.LISTING(DTTEST)`. The window that displays the source code comes up and you can start debugging the application.

The source of the active program is displayed in another window, shown in Figure 68 on page 79.

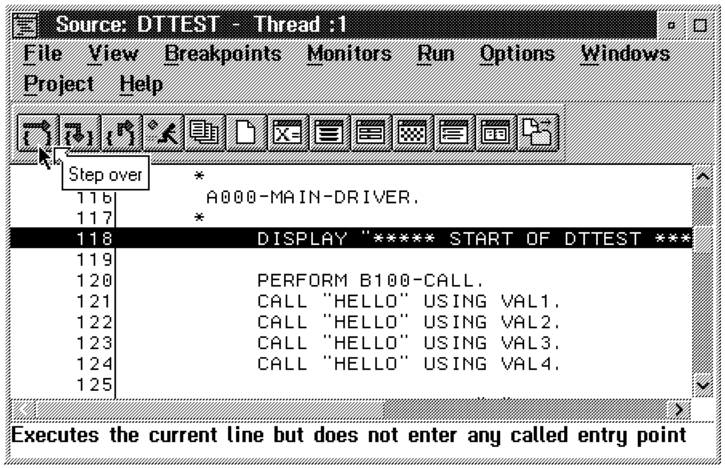

Source: DTTEST - Thread :1
File View Breakpoints Monitors Run Options Windows
Project Help

```
 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.     DTTEST.
 3        AUTHOR.         IBM PROGRAMMER.
 4        INSTALLATION.   STL
 5        DATE-WRITTEN.   MAY 25, 1977.
 6        DATE-COMPILED.  05/05/97.
 7
 8       ***************************************
 9       * LANGUAGE.                       VS COBOL
10       *
11       * TEST FUNCTION.
```

Figure 68. Source: DTTEST – Thread:1 Window

4. Click on the leftmost icon, called **Step over**, and you reach the line of the program ID where the program starts.

   Click again on this icon and you reach the first line of executable code, shown in Figure 69.


Source: DTTEST - Thread :1
File View Breakpoints Monitors Run Options Windows
Project Help
Step over

```
          *
116        A000-MAIN-DRIVER.
117       *
118            DISPLAY "***** START OF DTTEST ***
119
120            PERFORM B100-CALL.
121            CALL "HELLO" USING VAL1.
122            CALL "HELLO" USING VAL2.
123            CALL "HELLO" USING VAL3.
124            CALL "HELLO" USING VAL4.
125
```

Executes the current line but does not enter any called entry point

Figure 69. Source: DTTEST - Thread:1 Window – Start of DTTEST

5. Select the line number of the command called **PERFORM B100-CALL.** and double-click on it.

   The line number appears in red, and with this you have set a breakpoint.

6. Double-click in the same way on two of the line numbers with the **CALL HELLO...** and double-click to set breakpoints.

   With each set of a breakpoint, the **Debugger - Session Control** window changes its status from Ready to Busy for a moment.

7. Click on the icon with the small running man and you step to the first breakpoint, at the line **PERFORM B100-CALL**.

8. To debug this perform statement, click on the second icon on the icon list whose arrow directs down. This is the **Step into** icon to enter the called entry point. You can use it for debugging subprograms as well as subroutines.

   You reach the code being executed from this perform statement.

9. Your cursor highlights the line with the code MOVE 1111 TO CUST-NUM. Double-click on **CUST-NUM** to bring up the **Program Monitor** window (Figure 70). The value field for CUST-NUM is still empty.

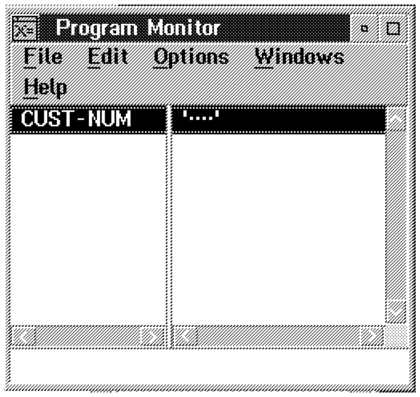


Figure 70. Program Monitor Window for DTTEST

10. Click again on **Step over** to reach the next line of code and the value for CUST-NUM changes to ′1111′. You see that the command is executed when the next command gets highlighted.

11. Double-click on **CUST-NAME** to display its value.

12. Execute the move command by executing **Step over** and the value for CUST-NAME changes to ′AAAAAAAAAA′.

13. Go to the Program Monitor window and double-click on ′**AAAAAAAAAA**′. You can change the value now. Change the value to ′**AAAAAAAABB**′ and press Enter to make it valid.

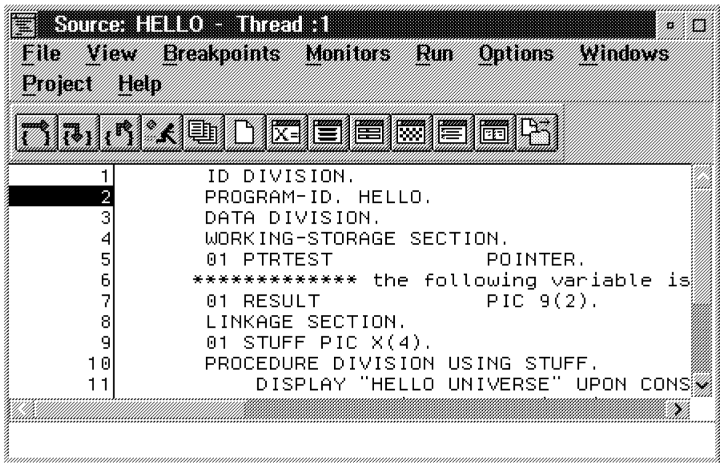
    Do not delete or overwrite the colons, because that invalidates the value. When you press Enter, the status of the **Debugger - Session Control** window changes to Busy for a moment. In the following execution the changed value of CUST-NAME is used.

14. Scroll down in the listing to line 155 where the command IF CUST-NUM NOT = 1111 ... is being executed and set a breakpoint here. Then click on the icon with the running man to reach this code.

15. Go further in the code using the **Step over** icon until you reach the line MOVE 3 TO CALLS-MADE and see how the values for the variables change in the **Program Monitor window**.

16. Then click on the running man icon to execute the following code through the next breakpoint. That is the first breakpoint of the CALL ″HELLO″ ... statements.

17. To debug this called program, click on the second icon from the left on the icon list whose arrow directs down. This is the icon **Step into** to enter the called entry point. The **HELLO - Thread 1** window shows you the source of the hello.cbl program now (Figure 71 on page 81).

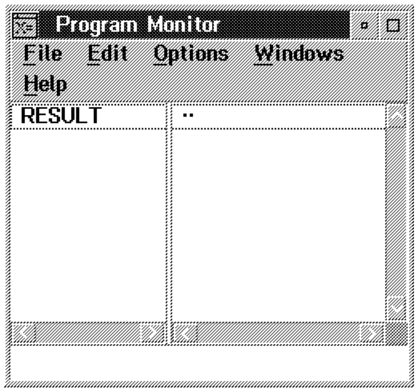

Source: HELLO - Thread :1
File View Breakpoints Monitors Run Options Windows
Project Help

```
      ID DIVISION.
      PROGRAM-ID. HELLO.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      01 PTRTEST          POINTER.
     ************* the following variable is
      01 RESULT           PIC 9(2).
      LINKAGE SECTION.
      01 STUFF PIC X(4).
      PROCEDURE DIVISION USING STUFF.
          DISPLAY "HELLO UNIVERSE" UPON CONS
```

Figure 71. Source: HELLO - Thread:1 Window

18. Click on the icon **Step over** to reach the first executable line of code. Scroll down in the source code to make the line COMPUTE RESULT EQUAL 5 * 4. visible.

19. Double-click on **RESULT** to bring up the **Program Monitor** window (Figure 72). The value for RESULT is still missing.



RESULT	..

Figure 72. Program Monitor Window for HELLO

20. Double-click on the **Step over** icon to reach the line DISPLAY ″5 * 4 = ″ RESULT UPON CONSOLE and you see that the value of RESULT has changed to 20.

21. Click on the third icon from the left on the icon list, **Step return**. It executes the rest of the program HELLO and stops at the statement that follows this CALL statement in the calling program. This is the next line with the command CALL ″HELLO″ ... you annotated with a breakpoint.

22. If you want, you can click on the **Step into** icon again to debug the hello subprogram. Otherwise, click again on the running man icon to end this debugger session.

    The last statement in the code is STOP-RUN. Because this is not the end of the file, the **Application Exception Action** window comes up and requests you to execute the exception handler (Figure 73 on page 82).

Click on **Run Exception** to leave this debugger session.

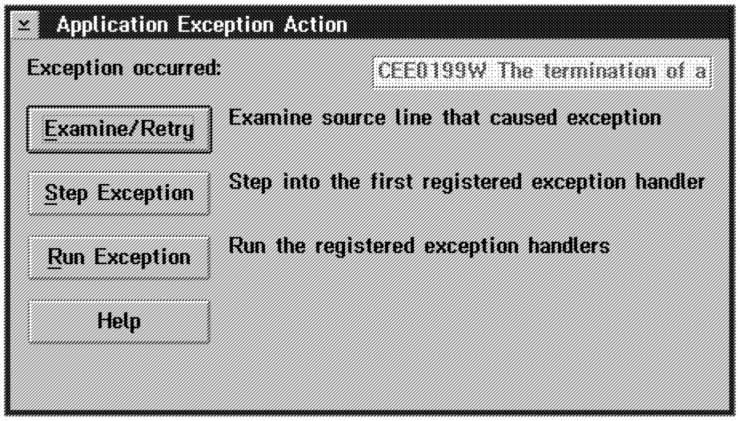

Application Exception Action
Exception occurred:
CEE0199W The termination of a
Examine/Retry
Examine source line that caused exception
Step Exception
Step into the first registered exception handler
Run Exception
Run the registered exception handlers
Help

Figure 73. Application Exception Action Window

23. The **Debugger Message** window informs you about the successful termination of the application (Figure 74). Click on **OK** to close the debugger.

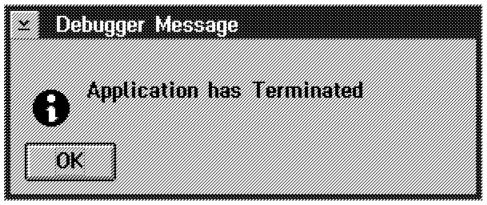


Figure 74. Debugger Message Window

Section 4.2, “Summary of Debugger Functionality and Additional Features” describes some additional features of the debug tool.

4.2 Summary of Debugger Functionality and Additional Features

As you have seen, you have many ways to step through the program. Depending on what you want to see, you can debug your code statement by statement using the Step over command, or you can define breakpoints in the code and use the command Run or the icon with the running man to reach the next statement on which you have selected a breakpoint before.

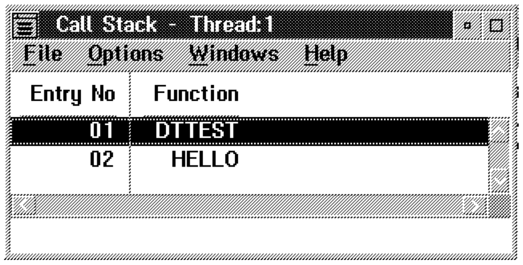
When you debug a program that calls subprograms, you can step into these subprograms by using the command Step into when the debug step reaches the statement that calls the subprogram. Otherwise, you step over the call and the code of the called program is executed but not displayed. Also, you have no chance to stop the debugging by setting breakpoints in this subprogram.

When you debug a subprogram, you can use the Step return command to execute the rest of the subprogram and jump back to the calling program.

To see the current values of variables and to be able to change them while the program executes, double-click on one of the variables that you want to see. The **Program Monitor** window comes up and displays the name of the variable and its value. Here, you can change the value by double-clicking on it; change it and press Enter. This is important, otherwise the debugger cannot recognize your entry. The debugging continues with the changed value for this variable when you click on one of the icons to let the program continue. The **Program Monitor** window shows you all the variables you want to see belonging to one program in one window. Their values change in the Program Monitor when they are changed by the program, so you always see their current values.

Further functions can help while debugging:

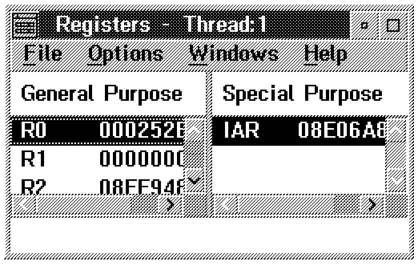
- The **Call Stack Thread:1** window can help you find out at which depth you are currently debugging when you have many nested programs (Figure 75).



Entry No	Function
01	DTTEST
02	HELLO

Figure 75. Call Stack Window

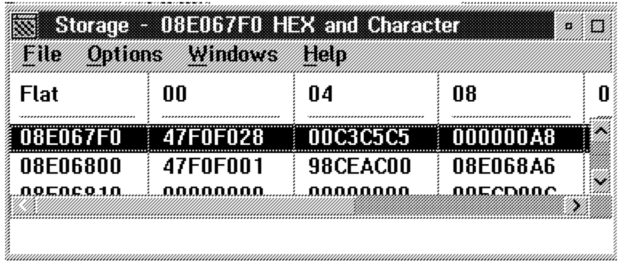
- The **Registers - Thread:1** window is useful to view or modify the contents of the registers monitored for your program (Figure 76).



```
R0    000252E        IAR    08E06A8
R1    000000C
R2    08FF94F
```

Figure 76. Registers Window

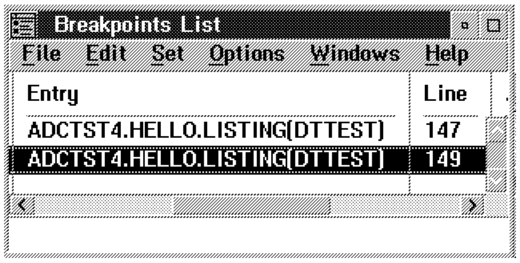
- Also the **Storage - 08E067F0 HEX and Character** window shows you values in hexadecimal and character format to view and update the contents of storage areas used by your program (Figure 77).


Storage - 08E067F0 HEX and Character
File Options Windows Help

Flat	00	04	08
08E067F0	47F0F028	00C3C5C5	000000A8
08E06800	47F0F001	98CEAC00	08E068A6

Figure 77. Storage Window

- Finally, the **Breakpoints List** window (Figure 78 on page 84) displays all your defined breakpoints in this program and gives you the choice to set, delete, enable, or disable each one.



Entry	Line
ADCTST4.HELLO.LISTING(DTTEST)	147
ADCTST4.HELLO.LISTING(DTTEST)	149

Figure 78. Breakpoints List Window

Breakpoints

Although the file that registers the breakpoints you set and the variables you trace is created, you always lose all these breakpoints and variables after program debugging. This is quite different from what happens in the workstation debugger.

As long as you are debugging the program, the session is still running on the host and nobody else can submit this job.

Connection error

If you start the debug session in WorkFrame and you get the error message "Unable to connect to remote debugger," then the debugger is already started and is waiting to connect to the host debugger. This is not really an error message.

Click on **OK** to leave this window and you can submit the job that runs the program with the debugger option.

Section 4.3, "DB2 Sample" presents a sample with DB2 access briefly described. Everything you have to do that is not described in each step should be known from the HELLO sample.

4.3 DB2 Sample

The first example we explained to you was a batch COBOL program. Now we want to build another sample program that accesses a DB2 database in addition.

You can also use the remote edit, compile, and debug feature for the maintenance of a DB2 batch program. The differences between these two examples are the following:

- In the new example, you have to create a database and we supply a JCL source file that can do that,
- You must add a DB2 precompile step before the compile step runs,
- You must use a different JCL to start the debugger because the system needs to execute the program under a TSO session.

- You must create a COBRS09.SAMPLEDB.DBRMLIB dataset. This dataset is necessary for compiling and executing our sample. To create such a dataset:
  - From a Prompt Command window, mount a local drive using the following command:
    ```
    mount u: wtsc47:"cobrs09,space(1,1),recfm(FB),lrecl(80),blksize(6160)"
    ```
  - Change to drive u: and create a new directory named SAMPLEDB.DBRMLIB with the following command:
    ```
    md sampledb.dbrmlib
    ```

The only important changes you have to make in these JCL members in order to use the Remote Editing, Compiling, Debugging feature are these:

- Use the TEST option for compiling your sources. For that, you only have to add TEST to the parameters.
- Use the /TEST(parameters) option for executing your load module to connect to the debugger.

  For TCP/IP, this parameter is the following where you specify your IP address:
  ```
  // REGION=32000K,PARM=('/TEST(,,,VADTCPIP&9.112.34.49:*)')
  ```
  For APPC, the REGION command varies as follows where you specify the machine name of your workstation:
  ```
  // REGION=32000K,PARM=('/TEST(,,,VADAPPC&STD1835:*)')
  ```

In order to create this sample, you first want to create the new database on the host. This is explained next.

4.3.1 Create a New Database on the Host

You have to create a new database on the host. For that a JCL is supplied on the diskette belonging to this book. This JCL contains structured query language (SQL) statements to create the database and tables, and to insert some rows into the tables.

In our example, we

1. Create a new database name
   ```
   CREATE DATABASE ITSODB99
   ```
2. Create a tablespace for this database
   ```
   CREATE TABLESPACE ITSOTS01 IN ITSODB99
   ```
3. Create two tables that we will use in our application:
   ```
   CREATE TABLE ITSOTB01
      (INVOICE     CHAR(05),
       CUSTOMER    CHAR(07),
       DATE        CHAR(06),
       REVENUE     DECIMAL (10,2));

   CREATE TABLE ITSOTB02
      (INVOICE     CHAR(05),
       PRODUCT     CHAR(07),
       AMOUNT      DECIMAL (3));
   ```

4. Give all users access to these tables:

```
GRANT ALL ON ITSOTB01 TO PUBLIC;

GRANT ALL ON ITSOTB02 TO PUBLIC;
```

5. Insert some rows in both tables:

```
INSERT INTO COBRS09.ITSOTB01 VALUES ('99309','0000102',
                                     '970328',2100);

INSERT INTO COBRS09.ITSOTB01 VALUES ('99308','0000295',
                                     '970328',1654);

INSERT INTO COBRS09.ITSOTB01 VALUES ('65202','0000345',
                                     '970329',65400);

INSERT INTO COBRS09.ITSOTB01 VALUES ('87224','0000372',
                                     '970330',5660);

INSERT INTO COBRS09.ITSOTB01 VALUES ('82394','0000146',
                                     '970330',6600);

INSERT INTO COBRS09.ITSOTB01 VALUES ('89634','0000231',
                                     '970331',66532);

INSERT INTO COBRS09.ITSOTB02 VALUES ('65202','TV00A02',50);

INSERT INTO COBRS09.ITSOTB02 VALUES ('65202','ST00A01',250);

INSERT INTO COBRS09.ITSOTB02 VALUES ('82394','VC00A01',300);

INSERT INTO COBRS09.ITSOTB02 VALUES ('89634','VC00A01', 50);

INSERT INTO COBRS09.ITSOTB02 VALUES ('89634','ST00A01',270);

INSERT INTO COBRS09.ITSOTB02 VALUES ('89634','TV00A02', 25);

INSERT INTO COBRS09.ITSOTB02 VALUES ('99309','ST00A01', 10);

INSERT INTO COBRS09.ITSOTB02 VALUES ('99308','ST00A01', 15);

INSERT INTO COBRS09.ITSOTB02 VALUES ('87224','VC00A01',150);
```

6. Finally, set up the COMMIT statement, which is requested for validating all the modifications that the system has received:

```
COMMIT.
```

In our example, we are using a sample database provided by DB2 for MVS. We have used the member DSN410.SDSNSAMP(DSNTIJTM). We are using DB2 Version 4.1. If you use a different version of DB2, modify the high-level qualifier of the member to find it on your host.

For using this JCL to create the database, you have to:

1. Copy this JCL with the same name provided by the DB2 for MVS into a data set with your user ID as high-level qualifier. Do not try to modify the original member.
2. Add your standard job card to the top of the JCL. In our example, the standard job card is the following:

```
//COBRS09C JOB (999,POK),NOTIFY=COBRS09,
//       CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
```

3. Remove all the lines up from the //DSNTIAS step to the end of the JCL. The JCL executes the following steps:

   a. It compiles and links an assembler procedure called DSNTIAD that is the utility for executing DB2 statements.

   b. It executes the DSNTIAD load module under TSO.

   For Step b, modify the parameters for the bind in the JCL. The part of the JCL that has to be modified is shown in Figure 79.

```
  ⋮
//DSNTIAB EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DSN)
  -ALTER BUFFERPOOL (BP0) VPSIZE(2000) HPSIZE(0) CASTOUT(YES)
  -ALTER BUFFERPOOL (BP32K) VPSIZE(24) HPSIZE(0) CASTOUT(YES)
//SYSIN    DD  *
//*
```

Figure 79. The Part of the JCL to Be Modified

4. Modify the DSN SYSTEM(DSN) line. You must specify the name of your DB2 system. We have changed our JCL to

```
  ⋮
     DSN SYSTEM(DB41)

  ⋮
```

   because the name of our DB2 system is DB41.

5. Replace the two -ALTER BUFFERPOOL lines with the lines that you need to execute the BIND PACKAGE, the BIND PLAN, and the RUN of the DSNTIAD load module. In our example, we have added the lines shown in Figure 30.

```
  ⋮
BIND PACKAGE(DSNTIAD) MEMBER(DSNTIAD) -
LIBRARY('DSN410.DBRMLIB.DATA') -
ACT(REP) ISOLATION(CS) VALIDATE(RUN) SQLERROR(CONTINUE)
BIND PLAN(DSNTIAD) PKLIST(DSNTIAD.*) -
ACT(REP) ISOLATION(CS) VALIDATE(RUN)
RUN PROGRAM(DSNTIAD)  PLAN(DSNTIAD) -
     LIB('DSN410.RUNLIB.LOAD')
END
  ⋮
```

Figure 80. Bind and Execute Statements of DSNTIAD

The hyphen at the end of some lines stands for the continuation of the command in the next line.

6. Add SQL commands right after the //SYSIN line to issue the creation of a new database.

The final part of your JCL should be similar to the following:

```
 :
    //DSNTIAB EXEC PGM=IKJEFT01,DYNAMNBR=20
    //SYSTSPRT DD  SYSOUT=*
    //SYSPRINT DD  SYSOUT=*
    //SYSUDUMP DD  SYSOUT=*
    //SYSTSIN  DD  *
      DSN SYSTEM(DB41)
      BIND PACKAGE(DSNTIAD) MEMBER(DSNTIAD) -
      LIBRARY('DSN410.DBRMLIB.DATA') -
      ACT(REP) ISOLATION(CS) VALIDATE(RUN) SQLERROR(CONTINUE)
      BIND PLAN(DSNTIAD) PKLIST(DSNTIAD.*) -
      ACT(REP) ISOLATION(CS) VALIDATE(RUN)
      RUN PROGRAM(DSNTIAD)  PLAN(DSNTIAD) -
           LIB('DSN410.RUNLIB.LOAD')
      END
    //SYSIN    DD  *

       DROP   TABLE    ITSOTB01;

       DROP   TABLE    ITSOTB02;

       DROP   DATABASE ITSODB99;

       COMMIT;

       CREATE DATABASE ITSODB99;

       CREATE TABLESPACE ITSOTS01 IN ITSODB99;

       CREATE TABLE ITSOTB01
        (INVOICE     CHAR(05),
         CUSTOMER    CHAR(07),
         DATE        CHAR(06),
         REVENUE     DECIMAL (10,2));

      GRANT ALL ON ITSOTB01 TO PUBLIC;

      CREATE TABLE ITSOTB02
       (INVOICE     CHAR(05),
        PRODUCT     CHAR(07),
        AMOUNT      DECIMAL (3));

      GRANT ALL ON ITSOTB02 TO PUBLIC;

      INSERT INTO COBRS09.ITSOTB01 VALUES ('99309','0000102',
                                           '970328',2100);

      INSERT INTO COBRS09.ITSOTB01 VALUES ('99308','0000295',
                                           '970328',1654);

      INSERT INTO COBRS09.ITSOTB01 VALUES ('65202','0000345',
                                           '970329',65400);
      INSERT INTO COBRS09.ITSOTB01 VALUES ('87224','0000372',
                                        '970330',5660);

      INSERT INTO COBRS09.ITSOTB01 VALUES ('82394','0000146',
                                        '970330',6600);

      INSERT INTO COBRS09.ITSOTB01 VALUES ('89634','0000231',
                                        '970331',66532);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('65202','TV00A02',50);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('65202','ST00A01',250);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('82394','VC00A01',300);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('89634','VC00A01', 50);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('89634','ST00A01',270);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('89634','TV00A02', 25);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('99309','ST00A01', 10);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('99308','ST00A01', 15);

      INSERT INTO COBRS09.ITSOTB02 VALUES ('87224','VC00A01',150);

      COMMIT;

//*
```

Now you can submit this job on the host. It compiles and links the assembler program DSNTIAD. Ensure that no error occurs in this submission.

The next step is to create the JCL for the DB2 precompile.

4.3.2 Create JCL Members to Compile the COBOL Program

We have already created a JCL that compiles and links a COBOL source without a DB2 precompile. To complete this JCL for the next application, do the following:

1. Provide the JCL with a DB2 precompiler step. This step will produce another COBOL source in which all DB2 statements are resolved by COBOL source. This is then a new input for your COBOL compiler that runs after this precompilation.

   A sample for this precompiler is the following code:

   ```
   //PC      EXEC PGM=DSNHPC,PARM='HOST(COB2)',REGION=4096K
   //DBRMLIB DD DSN=COBRS09.SAMPLEDB.DBRMLIB(&MEM),
   //         DISP=SHR
   //STEPLIB  DD  DISP=SHR,DSN=DSN410.SDSNEXIT
   //         DD  DISP=SHR,DSN=DSN410.SDSNLOAD
   //SYSCIN   DD  DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
   //             SPACE=(800,(&WSPC,&WSPC))
   //SYSLIB DD DSN=&USER..SRCLIB.DATA,
   //        DISP=SHR
   //SYSPRINT DD  SYSOUT=*
   //SYSTERM  DD  SYSOUT=*
   //SYSUDUMP DD  SYSOUT=*
   ```

```
//SYSUT1   DD  SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT2   DD  SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
```

2. Add a bind step after the link step. This is necessary to access the database. An example is shown here:

```
//PH02CS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD DSN=COBRS09.SAMPLEDB.DBRMLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//REPORT   DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DB41)
 BIND PACKAGE(COBRS09) MEMBER(SAMPLEDB) -
 LIBRARY('COBRS09.SAMPLEDB.DBRMLIB') -
 ACT(REP) ISOLATION(CS) VALIDATE(RUN) SQLERROR(CONTINUE)
 BIND PLAN(SAMPLEDB) PKLIST(COBRS09.*) -
 ACT(REP) ISOLATION(CS) VALIDATE(RUN)
 END
```

3. Provide the JCL with the TEST option in the parameter list to be able to debug the sample in the same way you did with the HELLO sample. This is necessary for using the remote debugger feature.

You cannot directly execute the load module from within the JCL because DB2 load files need to be started under TSO. So you have to add a parameter option similar to the following to your JCL:

```
DSN SYSTEM(DB41)
RUN  PROGRAM(SAMPLEDB) PLAN(SAMPLEDB) -
     LIB('COBRS09.SAMPLEDB.LOAD') -
     PARMS('/TEST(,,,VADTCPIP&9.112.32.49:*)')
END
```

for the connection with TCP/IP. The parameter list is different for the connection with APPC:

```
     PARMS('/TEST(,,,VADAPPC&STB1835:*)')
```

Now you can compile and run the program using the JCL. You can see that you have passed the /TEST(option) parameter in a different way, but this is nevertheless the parameter you have to use for connecting to the local debugger.

The diskette includes both JCL files. They are called DTBLDDB.JCL and DTRUNDB.JCL and are located in the path A:\SAMPLEDB\JCL.

To run the sample with DB2 access remotely, you have to create a WorkFrame project.

4.3.3 Creating the Project and Running the DB2 Sample

The WorkFrame project has to be created and the whole application has to be set up. To do so, follow these steps:

1. Create a Workframe project of the kind you need. Depending on whether you are connected using TCP/IP or using APPC, create a MVS project with or without W/S file extension.
2. In the project settings, specify the remote directory in which your sources should reside.

3. Create the data sets on the host you need for the COBOL source and the JCL members as well as for listings and load modules. Our data set for the COBOL source is called *COBRS09.SAMPLEDB.COBOL* and the data set for the JCL members is called *COBRS09.SAMPLEDB.JCL*, and so on.
4. Upload the files from the diskette to the host, or copy them remotely. You need all files located in the subdirectories of the directory A:\SAMPLEDB.
5. You have already set up your workstation for the previous example to connect to the host, and you have set up the remote directories that contain your sources. If you are working with the same host, the same user ID and with members contained in data sets that have the same high-level qualifier as in the previous example, you do not need to change your MVSINFO.DAT file. Otherwise, refer to the explanation given in Section 2.1.2, "MVSINFO.DAT File Explanation" on page 12.

The steps necessary to work with the project are now completed. Log on remotely on the host and make the files visible in the WorkFrame project.

You are now able to edit your sources, submit jobs for compiling and linking, and run the debugger.

When you debug the program, you see that all the DB2 statements in the code are in comments and instead of these, new COBOL statements get executed. The DB2 statements were replaced with COBOL statements by the DB2 precompiler.

The two examples we have shown you allow connection through either TCP/IP or APPC. The next example is executable only when you have set up the APPC and when configured with SMARTdata Utilities.

Chapter 5, "MVS Data Types Sample" on page 93 describes the use of remote VSAM files that are located on the host. This sample uses functions provided by SMARTdata Utilities to create a VSAM file on the host and to access it from the workstation.

Chapter 5. MVS Data Types Sample

This chapter shows you, by means of a sample, how to use VSAM for the workstation for local and remote data access, including customizing data description and conversion for transparent remote data access.

We describe an SdU sample that creates a VSAM file on the host first, and then lets you change single rows of this file. You can add rows and delete them, also.

This sample illustrates an application that accesses data from a VSAM file on a remote MVS system. The application provides options for creating and reading the file as well as updating or deleting a specific record.

The sample demonstrates the use of COBOL compile options CHAR(EBCDIC), FLOAT(HEX) and BINARY(S390) to process remote MVS data from the workstation without any changes to program logic. It also demonstrates remote file access via APPC on OS/2 platforms and creating and accessing a remote VSAM data set from a workstation without executing any VSAM utilities on the remote system.

Because the sample is provided by VisualAge for COBOL, you do not have to create a new project. Instead, you can find the project in the Samples folder.

To build the sample, repeat these steps:

1. Double-click on the **VisualAge for COBOL** icon on the Desktop.
2. Double-click on the **Samples** folder.
3. Double-click on the **MVS Data Types Sample** project. The WorkFrame with the **MVS Data Types - Icon view** window comes up (Figure 81).

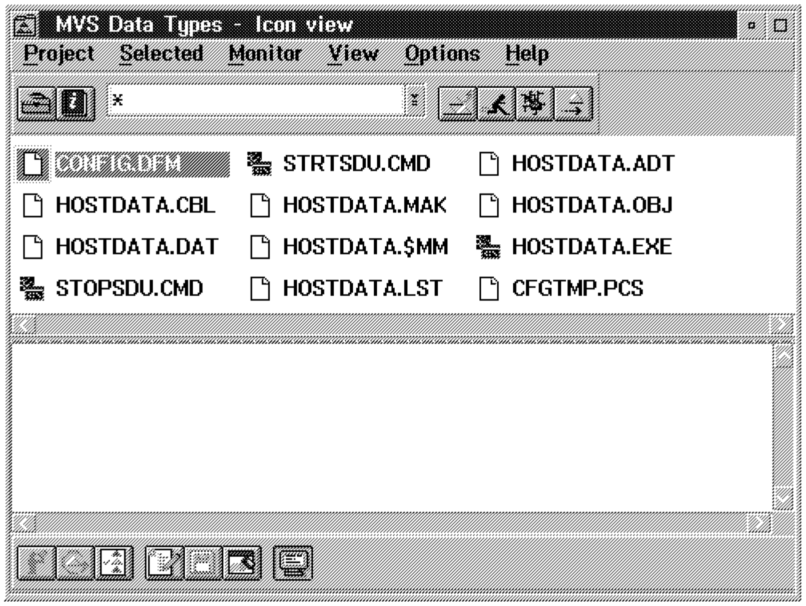


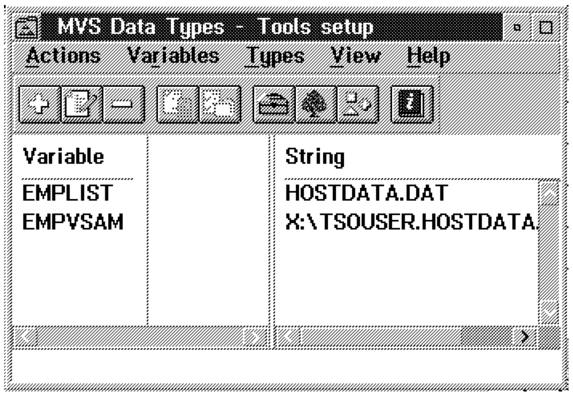
Figure 81. MVS Data Type - Icon view Window

4. In the project, click on the **Build** icon in the tool bar. The output from the build operation is displayed in the project monitor area.

To create a VSAM file on your MVS system, configure the sample in the following way:

- Either copy your CONFIG.DFM file (which you have already updated with your specifications as described in 3.2.1, “Configuration of SMARTdata Utilities on the Workstation” on page 53) or update the supplied CONFIG.DFM file for your site.

  Your CONFIG.DFM file is located in the path D:\IBMCOBOL\MACROS (D: is the drive where your VisualAge for COBOL is installed). To update the provided CONFIG.DFM file in this project, follow the steps in Section 3.2.1, “Configuration of SMARTdata Utilities on the Workstation” on page 53.
- Update the supplied STRTSDU.CMD file by changing the drive letter that represents your remote system from J to a letter compatible with the workstation user′s drive letter assignments. We used M.
- Then update the STOPSDU.CMD file in the same way and change the drive letter from J to the letter you specified in STRTSDU.CMD.
- Use the Tools Setup option to change the value for the variable named EMPVSAM, which is used to name the VSAM file to be created. To do so, select **View** in the menu bar and then **Tools setup** in the pull-down menu, or just click on the icon with the small suitcase on the WorkFrame. The **MVS Data Types - Tools setup** window comes up (Figure 82).



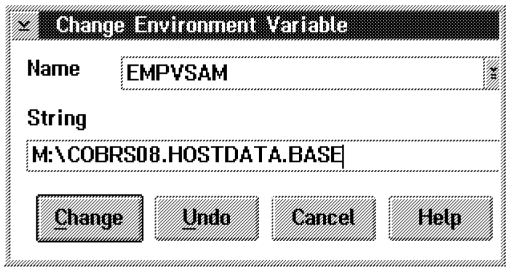
Variable	String
EMPLIST	HOSTDATA.DAT
EMPVSAM	X:\TSOUSER.HOSTDATA.

Figure 82. MVS Data Types - Tools setup Window

Double-click on the variable EMPVSAM and the **Change Environment Variable** window comes up, in which you can change the drive letter and at least the high-level qualifier for the data base to be created, as shown in Figure 83 on page 95.

Replace the supplied value x:\tsouser.hostdata.base with a drive letter and data set name appropriate for your site:

- Set x to the drive letter representing the remote system—in our case, M.
- Set TSOUSER to your user ID on your remote system—in our case, COBRS08.



Name: EMPVSAM

String: M:\COBRS08.HOSTDATA.BASE

Figure 83. Change Environment Variable Window

Click on **Change** to make the changes valid and then close the **MVS Data Type - Tools setup** window by double-clicking on the icon in the upper left corner of the window.

Authorization

The TSOUSER must be one who is authorized to create VSAM data sets on your remote system. The flat file input (hostdata.dat) supplied for creating the VSAM file is specified with the EMPLIST environment variable.

- Start the SMARTdata Utilities using the STRTSDU.CMD file. To do so, double-click on it or select the **STRTSDU.CMD** file and request the pop-up menu using mouse button 2, then choose **RUN**. Reply to the prompt for password and again to close the window.
- Run the HOSTDATA.EXE program by clicking on the **Run** icon in the tool bar.

The system displays the message:

```
Create or Maintain a VSAM employee file.

Enter a choice:
 1 to Create
 2 to Report
 3 to Update a single record
 4 to Delete a single record
```

Use Option 1 (create) to load the data.

```
Opening for CREATE

Starting VSAM file loader...
Using EMPVSAM variable for name of output file
..... EMPLIST for name of input file.
SYSIN file opened OK using EMPLIST variable
OUTPUT file opened OK using EMPVSAM variable
Loading....DOE          BILL      DD35 01 1234643210
Loading....DOE          JANE      AA55 02 1234743210
Loading....DOE          JOHN      BB41 03 1234843210
Loading....DOWE         SUE       TD35 04 1234943210
Loading....JOHNSON      JANE      TB44 05 1235043210
Loading....JONES        BILL      NA55 06 1235143210
Loading....JONES        BOB       FB41 07 1235243210
Loading....JONES        MARY      CD35 08 1235343210
Loading....SMITH        BOB       MB44 09 1235443210
Loading....SMITH        MARY      SA55 10 1235543210
Loading....SMITH          SUE       RB41 11 1235643210
Loading....SMITHE         BILL      EA55 12 1235743210

After close of OUTPUT file:
Status: 00 VSAM-RC=0000 VSAM-FUNC=0000 VSAM-FEED=0000

Reopen the OUTPUT file to verify it's usable

OUTPUT file is ok, file open status =00
First record=DOE

Press enter to exit
```

If you get an error message while creating the VSAM file, make sure that the drive representing the remote system is the same in STRTSDU.CMD, STOPSDU.CMD and in the tools setup where you specified the data set name. If not, the error message you get looks like this:

```
1
Opening for CREATE

Starting VSAM file loader...
Using EMPVSAM variable for name of output file
..... EMPLIST for name of input file.
SYSIN file opened OK using EMPLIST variable
Error opening OUTPUT file. Status code = 91
Verify your environment variable points to a valid remote drive.

Press enter to exit
```

Once the VSAM file is created on your MVS system, you can run the HOSTDATA.EXE program by double-clicking on it or by clicking on the **Run** icon in the tool bar.

As it runs,

1. Verify that the remote VSAM file data is being read, converted, and displayed properly using Option 2 (Report). A report is being created:

```
Using EMPVSAM variable...for name of employee file.
Opening for REPORT

Employee-Name              Dept Phone        Hired
-------------------------- ---  ------------ ------
DOE            BILL      D D35  408-555-9995 781206
        01 0223 1234643210
         .41399994E 01
         .41456700000000000E 01
        01 1087 +1234567954321
DOE            JANE      A A55  212-555-9950 890726
        02 0323 1234743210
         .51399994E 01
         .51456700000000000E 01
        02 1187 +1234568054321


     ⋮
```

```
SMITH          SUE       R B41  202-555-8989 740701
        11 1223 1235643210
         .14140000E 02
         .1414567000000000E 02
        11 2087 +1234568954321
SMITHE         BILL      E A55  212-555-7535 821229
        12 1323 1235743210
         .15140000E 02
         .1514567000000000E 02
        12 2187 +1234569054321

Reading done -- last read status code=10

Press enter to exit
```

To stop the listing of the report and be able to read it, press the Pause key, and to release it again press Enter.

2. Update a record using Option 3 (Update).

   With the update function, you can either update a row of this VSAM file or add a new row. When you select Option 3 for update, you are prompted to give the employee name in three parts. First enter the last name, then the first name, and then the middle initial, typing it all in upper case letters.

   If the VSAM file does not contain this name, the system tells you

```
Status code=23 - Record not found
Adding...
```

   and wants to insert a new employee to the list. It requests you to enter also a department number, hire date, and phone number. After doing so, you get the confusing message:

```
Status code=00 - update complete.

Press enter to exit
```

   but the row is inserted and with the next report you also get this added row.

3. Delete a record using Option 4 (Delete).

   To delete a row of the VSAM file, use Option 4. The system prompts you to insert the name in three parts in the same way as you did the update.

   After a successful deletion, you get the message referring to the name to be deleted, for example like this:

```
Status code=00 - Record found
Deleting...JONES          BILL      N
Status code=00 - Record deleted

Press enter to exit
```

   If the record is not found, the system comes up with status code=23.

Create the VSAM file twice

You can create the VSAM file again under the same name without getting an error message. The old file is overwritten with the original data.

To leave this application, run STOPSDU.CMD to release the attached drive.

Appendix A. DFM/MVS DataAgent

The DFM/MVS DataAgent extends the function of the Distributed FileManager (DFM) component of DFSMSdfp by providing the ability to invoke routines that run as extensions of DFM/MVS from remote SMARTdata Utilities (SdU) clients. By providing for the ability to add routines that can be invoked by DFM/MVS clients, this new capability brings a new dimension to the way client applications can access data on MVS/ESA or OS/390 beyond predefined remote transactions.

Until now, the workstation application used data management commands to operate directly on the remote data using DFM/MVS. Now this mode of operation has been coupled with the ability to bracket the workstation application′s execution by providing for the execution of jobs on the remote MVS system. There are three phases in this new scenario. The preprocessing phase occurs just before the workstation application begins to execute. The execution phase is the execution of the workstation application. The postprocessing phase occurs immediately after the workstation application ends its processing. The new execution sequence is controlled by extensions to commands currently used by the client (SMARTdata Utilities) to communicate with the DFM/MVS server.

Additional information on the DataAgent is available in DFSMS/MVS Version 1 Release 4 Technical Guide, SG24-4892 and the DFSMS/MVS Version 1 Release 4 Distributed FileManager/MVS Guide and Reference, SC26-4915-02.

Appendix B. Contents of the Enclosed Diskette

On the diskette you will find the following:

```
The directory, CONNECT, includes the following files:

   MVSINFO.TCP as a sample for the MVSINFO.DAT file
               for the connection with TCP/IP
   MVSINFO.APC as a sample for the MVSINFO.DAT file
               for the connection with APPC and SdU
    CONFIG.DFM as a sample for the CONFIG.DFM file

The directory, HELLO, includes all necessary files for the HELLO
sample as it is described in the redbook

The directory, SAMPLEDB, includes all necessary files for the
SAMPLEDB samples as it is described in the redbook.
```

Appendix C. Special Notices

This publication is intended to help COBOL application developers who want to develop host-based COBOL applications using features and functions on the workstation. The information in this publication is not intended as the specification of any programming interfaces that are provided by VisualAge for COBOL on OS/2 Version 2.0 (Standard Edition). See the PUBLICATIONS section of the IBM Programming Announcement for VisualAge for COBOL for OS/2 Version 2.0 (Standard Edition) for more information about what publications are considered to be product documentation.

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 IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

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 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

Advanced Peer-to-Peer Networking®	DATABASE 2
DB2®	DFSMS
DFSMS/MVS®	DFSMShsm
IBM®	MVS® (logo)
MVS/ESA	Operating System/2®
OS/2®	OS/390
S/390®	VisualAge®

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 107.

- *Host/Workstation Client/Server Implementation Using VisualAge COBOL on OS/2, AIX, and MVS*, SG24-4733
- *IBM VisualAge for COBOL for COBOL for OS/2 OO Programming*, SG24-4606

D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	**Subscription Number**	**Collection Kit Number**
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

D.3 Other Publications

These publications are also relevant as further information sources:

- *VisualAge for COBOL Language Reference*, SC26-9046
- *VisualAge for COBOL Programming Guide*, SC26-9050
- *VisualAge for COBOL User′s Guide for OS/2*, SC26-9036
- *SMARTdata Utilities for OS/2 - VSAM in a Distributed Environment*, SC26-7063
- *SMARTdata Utilities - Data Description and Conversion*, SC26-7091
- *SMARTdata Utilities - Data Description - A Data Language Reference*, SC26-7092

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type `GOPHER.WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at ″Redpieces″ on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.htm`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at ″Redpieces″ on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name　　Last name

Company

Address

City　　Postal code　　Country

Telephone number　　Telefax number　　VAT number

- Invoice to customer number

- Credit card number

Credit card expiration date　　Card issued to　　Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms may or may not have the same meaning in other languages.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the following publications:

- *American National Standard Programming Language COBOL, ANSI X3.23-1985* (Copyright 1985 American National Standards Institute, Inc.), which was prepared by Technical Committee X3J4, which had the task of revising American National Standard COBOL, X3.23-1974.
- *American National Dictionary for Information Processing Systems* (Copyright 1982 by the Computer and Business Equipment Manufacturers Association).

American National Standard definitions are preceded by an asterisk (*).

A

*** abbreviated combined relation condition**. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

abend. Abnormal termination of program.

*** access mode**. The manner in which records are to be operated upon within a file.

*** actual decimal point**. The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

*** alphabet-name**. A user-defined word, in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, that assigns a name to a specific character set and/or collating sequence.

*** alphabetic character**. A letter or a space character.

*** alphanumeric character**. Any character in the computer's character set.

alphanumeric-edited character. A character within an alphanumeric character-string that contains at least one B, 0 (zero), or / (slash).

*** alphanumeric function**. A function whose value is composed of a string of one or more characters from the computer′s character set.

*** alternate record key**. A key, other than the prime record key, whose contents identify a record within an indexed file.

ANSI (American National Standards Institute). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

*** argument**. An identifier, a literal, an arithmetic expression, or a function-identifier that specifies a value to be used in the evaluation of a function.

*** arithmetic expression**. An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

*** arithmetic operation**. The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

*** arithmetic operator**. A single character, or a fixed two-character combination that belongs to the following set:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

*** arithmetic statement**. A statement that causes an arithmetic operation to be executed. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements.

array. In Language Environment, an aggregate consisting of data objects, each of which may be uniquely referenced by subscripting. Roughly analogous to a COBOL table.

*** ascending key**. A key upon the values of which data is ordered, starting with the lowest value of the key up to the highest value of the key, in accordance with the rules for comparing data items.

ASCII. American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange between data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

Extension: IBM has defined an extension to ASCII code (characters 128-255).

assignment-name. A name that identifies the organization of a COBOL file and the name by which it is known to the system.

*** assumed decimal point**. A decimal point position that does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning with no physical representation.

*** AT END condition**. A condition caused:

1. During the execution of a READ statement for a sequentially accessed file, when no next logical record exists in the file, or when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional input file is not present.
2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.
3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

B

big-endian. Default format used by the mainframe and the AIX workstation to store binary data. In this format, the least significant digit is on the highest address. Compare with "little-endian."

binary item. A numeric data item represented in binary notation (on the base 2 numbering system). Binary items have a decimal equivalent consisting of the decimal digits 0 through 9, plus an operational sign. The leftmost bit of the item is the operational sign.

binary search. A dichotomizing search in which, at each step of the search, the set of data elements is divided by two; some appropriate action is taken in the case of an odd number.

*** block**. A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

breakpoint. A place in a computer program, usually specified by an instruction, where its execution may be interrupted by external intervention or by a monitor program.

Btrieve. A key-indexed record management system that allows applications to manage records by key value, sequential access method, or random access method. IBM COBOL supports COBOL sequential and indexed file I-O language through Btrieve.

buffer. A portion of storage used to hold input or output data temporarily.

built-in function. See "intrinsic function."

byte. A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character.

C

callable services. In Language Environment, a set of services that can be invoked by a COBOL program using the conventional Language Environment-defined call interface, and usable by all programs sharing the Language Environment conventions.

called program. A program that is the object of a CALL statement.

*** calling program**. A program that executes a CALL to another program.

case structure. A program processing logic in which a series of conditions is tested in order to make a choice between a number of resulting actions.

cataloged procedure. A set of job control statements placed in a partitioned data set called the procedure library (SYS1.PROCLIB). You can use cataloged procedures to save time and reduce errors coding JCL.

century window. The 100-year interval in which Language Environment assumes all 2-digit years lie. The Language Environment default century window begins 80 years before the system date.

*** character**. The basic indivisible unit of the language.

character position. The amount of physical storage required to store a single standard data format character described as USAGE IS DISPLAY.

character set. All the valid characters for a programming language or a computer system.

*** character-string**. A sequence of contiguous characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry. Must be delimited by separators.

checkpoint. A point at which information about the status of a job and the system can be recorded so that the job step can be later restarted.

*** 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.

*** class condition**. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic, is wholly numeric, or consists exclusively of those characters listed in the definition of a class-name.

*** Class Definition**. The COBOL source unit that defines a class.

*** class identification entry**. An entry in the CLASS-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the class-name and assign selected attributes to the class definition.

*** class-name**. A user-defined word defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION that assigns a name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

class object. The run-time object representing a SOM class.

*** clause**. An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

CMS (Conversational Monitor System). A virtual machine operating system that provides general interactive, time-sharing, problem solving, and program development capabilities, and that operates only under the control of the VM/SP control program.

*** COBOL character set**. The complete COBOL character set consists of the characters listed below:

Character	**Meaning**
0,1...,9	digit
A,B,...,Z	uppercase letter
a,b,...,z	lowercase letter
•	space
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	slant (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
″	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

*** COBOL word**. See “word.”

code page. An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for 8-bit code, assignment of characters and meanings to 128 code points for 7-bit code.

*** collating sequence**. The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, comparing, and for processing indexed files sequentially.

*** column**. A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

*** combined condition**. A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

*** comment-entry**. An entry in the IDENTIFICATION DIVISION that may be any combination of characters from the computer’s character set.

*** comment line**. A source program line represented by an asterisk (*) in the indicator area of the line and any characters from the computer’s character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a slant (/) in the indicator area of the line and any characters from the computer’s character set in area A and area B of that line causes page ejection prior to printing the comment.

*** common program**. A program which, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

*** compile**. (1) To translate a program expressed in a high-level language into a program expressed in an intermediate language, assembly language, or a computer language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler.

*** compile time**. The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

compiler. A program that translates a program written in a higher level language into a machine language object program.

compiler directing statement. A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

compiler directing statement. A statement that specifies actions to be taken by the compiler during processing of a COBOL source program. Compiler directives are contained in the COBOL source program. Thus, you can specify different suboptions of the directive within the source program by using multiple compiler directive statements in the program.

*** complex condition**. A condition in which one or more logical operators act upon one or more conditions. (See also “negated simple condition,” “combined condition,” and “negated combined condition.”)

*** computer-name**. A system-name that identifies the computer upon which the program is to be compiled or run.

condition. An exception that has been enabled, or recognized, by Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and results in an interrupt. They can also be detected by language-specific generated code or language library code.

*** condition**. A status of a program at run time for which a truth value can be determined. Where the term ‘condition’ (condition-1, condition-2,...) appears in these language specifications in or in reference to ‘condition’ (condition-1, condition-2,...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

*** conditional expression**. A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM, or SEARCH statement. (See also “simple condition” and “complex condition.”)

*** conditional phrase**. A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

*** conditional statement**. A statement specifying that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

*** conditional variable**. A data item one or more values of which has a condition-name assigned to it.

*** condition-name**. A user-defined word that assigns a name to a subset of values that a conditional variable may assume; or a user-defined word assigned to a status of an implementor defined switch or device. When ‘condition-name’ is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a ‘condition-name’, together with qualifiers and subscripts, as required for uniqueness of reference.

*** condition-name condition**. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

*** CONFIGURATION SECTION**. A section of the ENVIRONMENT DIVISION that describes overall specifications of source and object programs and class definitions.

CONSOLE. A COBOL environment-name associated with the operator console.

*** contiguous items**. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

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

CORBA. The Common Object Request Broker Architecture established by the Object Management Group. IBM′s *Interface Definition Language* used to describe the *interface* for SOM classes is fully compliant with CORBA standards.

*** counter**. A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

cross-reference listing. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

currency sign. The character ‘$’ of the COBOL character set or that character defined by the CURRENCY compiler option. If the NOCURRENCY compiler option is in effect, the currency sign is defined as the character ‘$’.

currency symbol. The character defined by the CURRENCY compiler option or by the CURRENCY

SIGN clause in the SPECIAL-NAMES paragraph. If the NOCURRENCY compiler option is in effect for a COBOL source program and the CURRENCY SIGN clause is also **not** present in the source program, the currency symbol is identical to the currency sign.

*** current record**. In file processing, the record that is available in the record area associated with a file.

*** current volume pointer**. A conceptual entity that points to the current volume of a sequential file.

D

*** data clause**. A clause, appearing in a data description entry in the DATA DIVISION of a COBOL program, that provides information describing a particular attribute of a data item.

*** data description entry** . An entry in the DATA DIVISION of a COBOL program that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

DATA DIVISION. One of the four main components of a COBOL program, class definition, or method definition. The DATA DIVISION describes the data to be processed by the object program, class, or method: files to be used and the records contained within them; internal working-storage records that will be needed; data to be made available in more than one program in the COBOL run unit. (Note, the Class DATA DIVISION contains only the WORKING-STORAGE SECTION.)

*** data item**. A unit of data (excluding literals) defined by a COBOL program or by the rules for function evaluation.

*** data-name**. A user-defined word that names a data item described in a data description entry. When used in the general formats, ‘data-name’ represents a word that must not be reference-modified, subscripted or qualified unless specifically permitted by the rules for the format.

DBCS (Double-Byte Character Set). See “Double-Byte Character Set (DBCS).”

*** debugging line**. A debugging line is any line with a ‘D’ in the indicator area of the line.

*** debugging section**. A section that contains a USE FOR DEBUGGING statement.

*** declarative sentence**. A compiler directing sentence consisting of a single USE statement terminated by the separator period.

*** declaratives**. A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one, or more associated paragraphs.

*** de-edit**. The logical removal of all editing characters from a numeric edited data item in order to determine that item′s unedited numeric value.

*** delimited scope statement**. Any statement that includes its explicit scope terminator.

*** delimiter**. A character or a sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

*** descending key**. A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

digit. Any of the numerals from 0 through 9. In COBOL, the term is not used in reference to any other symbol.

*** digit position**. The amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the data description entry that defines the data item.

*** direct access**. The facility to obtain data from storage devices or to enter data into a storage device in such a way that the process depends only on the location of that data and not on a reference to data previously accessed.

*** division**. A collection of zero, one or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

*** division header**. A combination of words followed by a separator period that indicates the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

DLL. See “dynamic link library.”

do construction. In structured programming, a DO statement is used to group a number of statements in a procedure. In COBOL, an in-line PERFORM statement functions in the same way.

do-until. In structured programming, a do-until loop will be executed at least once, and until a given condition is true. In COBOL, a TEST AFTER phrase used with the PERFORM statement functions in the same way.

do-while. In structured programming, a do-while loop will be executed if, and while, a given condition is true. In COBOL, a TEST BEFORE phrase used with the PERFORM statement functions in the same way.

Double-Byte Character Set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require Double-Byte Character Sets. Because each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

*** dynamic access**. An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

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.

Dynamic Storage Area (DSA). Dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation (such as program variables). DSAs are generally allocated within STACK segments managed by Language Environment.

E

*** EBCDIC (Extended Binary-Coded Decimal Interchange Code)**. A coded character set consisting of 8-bit coded characters.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.

edited data item. A data item that has been modified by suppressing zeroes and/or inserting editing characters.

*** editing character**. A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
•	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	slant (virgule, slash)

element (text element). One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

*** elementary item**. A data item that is described as not being further logically subdivided.

enclave. When running under the Language Environment product, an enclave is analogous to a run unit. An enclave can create other enclaves on OS/390 and CMS by a LINK, on CMS by CMSCALL, and the use of the system () function of C.

***end class header**. A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class header is:

```
END CLASS class-name.
```

***end method header**. A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method header is:

```
END METHOD method-name.
```

*** end of Procedure Division**. The physical position of a COBOL source program after which no further procedures appear.

*** end program header**. A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program header is:

```
END PROGRAM program-name.
```

*** entry**. Any descriptive set of consecutive clauses terminated by a separator period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL program.

*** environment clause**. A clause that appears as part of an ENVIRONMENT DIVISION entry.

ENVIRONMENT DIVISION. One of the four main component parts of a COBOL program, class definition, or method definition. The ENVIRONMENT DIVISION describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

environment-name. A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, and/or program switches. When an environment-name is associated with a mnemonic-name in the ENVIRONMENT DIVISION, the mnemonic-name may then be substituted in any format in which such substitution is valid.

environment variable. Any of a number of variables that describe the way an operating system is going to run and the devices it is going to recognize.

execution time. See “run time.”

execution-time environment. See “run-time environment.”

*** explicit scope terminator**. A reserved word that terminates the scope of a particular Procedure Division statement.

exponent. A number, indicating the power to which another number (the base) is to be raised. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, an exponential expression is indicated with the symbol ‘**’ followed by the exponent.

*** expression**. An arithmetic or conditional expression.

*** extend mode**. The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file, and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

extensions. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

*** external data**. The data described in a program as external data items and external file connectors.

*** external data item**. A data item which is described as part of an external record in one or more programs of a run unit and which itself may be referenced from any program in which it is described.

*** external data record**. A logical record which is described in one or more programs of a run unit and whose constituent data items may be referenced from any program in which they are described.

external decimal item. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte. Bits 0 through 3 of all other bytes contain 1’s (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. (Also know as “zoned decimal item.”)

*** external file connector**. A file connector which is accessible to one or more object programs in the run unit.

external floating-point item. A format for representing numbers in which a real number is represented by a pair of distinct numerals. In a floating-point representation, the real number is the product of the fixed-point part (the first numeral), and a value obtained by raising the implicit floating-point base to a power denoted by the exponent (the second numeral).

For example, a floating-point representation of the number 0.0001234 is: 0.1234 -3, where 0.1234 is the mantissa and -3 is the exponent.

*** external switch**. A hardware or software device, defined and named by the implementor, which is used to indicate that one of two alternate states exists.

F

*** figurative constant**. A compiler-generated value referenced through the use of certain reserved words.

*** file**. A collection of logical records.

*** file attribute conflict condition**. An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

*** file clause**. A clause that appears as part of any of the following DATA DIVISION entries: file description entry (FD entry) and sort-merge file description entry (SD entry).

*** file connector**. A storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

File-Control. The name of an ENVIRONMENT DIVISION paragraph in which the data files for a given source program are declared.

*** file control entry**. A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

*** file description entry**. An entry in the File Section of the DATA DIVISION that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

*** file-name**. A user-defined word that names a file connector described in a file description entry or a sort-merge file description entry within the File Section of the DATA DIVISION.

*** file organization**. The permanent logical file structure established at the time that a file is created.

***file position indicator**. A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that an optional input file is not present, or that the at end condition already exists, or that no valid next record has been established.

*** File Section**. The section of the DATA DIVISION that contains file description entries and sort-merge file description entries together with their associated record descriptions.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

*** fixed file attributes**. Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

*** fixed length record**. A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

fixed-point number. A numeric data item defined with a PICTURE clause that specifies the location of an optional sign, the number of digits it contains, and the location of an optional decimal point. The format may be either binary, packed decimal, or external decimal.

floating-point number. A numeric data item containing a fraction and an exponent. Its value is obtained by multiplying the fraction by the base of the numeric data item raised to the power specified by the exponent.

*** format**. A specific arrangement of a set of data.

*** function**. A temporary data item whose value is determined at the time the function is referenced during the execution of a statement.

*** function-identifier**. A syntactically correct combination of character-strings and separators that references a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function-identifier may include a reference-modifier. A function-identifier that references an alphanumeric function may be specified anywhere in the general formats that an identifier may be specified, subject to certain restrictions. A function-identifier that references an integer or numeric function may be referenced anywhere in the general formats that an arithmetic expression may be specified.

function-name. A word that names the mechanism whose invocation, along with required arguments, determines the value of a function.

G

*** global name**. A name which is declared in only one program but which may be referenced from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, report-names, and some special registers may be global names.

*** group item**. A data item that is composed of subordinate data items.

H

header label. (1) A file label or data set label that precedes the data records on a unit of recording media. (2) Synonym for beginning-of-file label.

*** high order end**. The leftmost character of a string of characters.

I

IBM COBOL extension. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

IDENTIFICATION DIVISION. One of the four main component parts of a COBOL program, class definition, or method definition. The IDENTIFICATION DIVISION identifies the program name, class name, or method name. The IDENTIFICATION DIVISION may include the following documentation: author name, installation, or date.

*** identifier**. A syntactically correct combination of character-strings and separators that names a data item. When referencing a data item that is not a function, an identifier consists of a data-name, together with its qualifiers, subscripts, and reference-modifier, as required for uniqueness of reference. When referencing a data item which is a function, a function-identifier is used.

IGZCBSN. The COBOL/370 Release 1 bootstrap routine. It must be link-edited with any module that contains a COBOL/370 Release 1 program.

IGZCBSO. The COBOL for MVS & VM Release 2 and IBM COBOL for OS/390 & VM bootstrap routine. It must be link-edited with any module that contains a COBOL for MVS & VM Release 2 or IBM COBOL for OS/390 & VM program.

*** imperative statement**. A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

*** implicit scope terminator**. A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

*** index**. A computer storage area or register, the content of which represents the identification of a particular element in a table.

*** index data item**. A data item in which the values associated with an index-name can be stored in a form specified by the implementor.

indexed data-name. An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

*** indexed file**. A file with indexed organization.

*** indexed organization**. The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

indexing. Synonymous with subscripting using index-names.

*** index-name**. A user-defined word that names an index associated with a specific table.

*** inheritance (for classes)**. A mechanism for using the implementation of one or more *classes* as the basis for another class. A *sub-class* inherits from one or more *super-classes*. By definition the inheriting class conforms to the inherited classes.

*** initial program**. A program that is placed into an initial state every time the program is called in a run unit.

*** initial state**. The state of a program when it is first called in a run unit.

inline. In a program, instructions that are executed sequentially, without branching to routines, subroutines, or other programs.

*** input file**. A file that is opened in the INPUT mode.

*** input mode**. The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

*** input-output file**. A file that is opened in the I-O mode.

*** INPUT-OUTPUT SECTION**. The section of the ENVIRONMENT DIVISION that names the files and the external media required by an object program or method and that provides information required for transmission and handling of data during execution of the object program or method definition.

*** Input-Output statement**. A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT (with the identifier phrase), CLOSE, DELETE, DISPLAY, OPEN, READ, REWRITE, SET (with the TO ON or TO OFF phrase), START, and WRITE.

*** input procedure**. A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

instance data. Data defining the state of an object. The instance data introduced by a class is defined in the WORKING-STORAGE SECTION of the DATA DIVISION of the class definition. The state of an object also includes the state of the instance variables introduced by base classes that are inherited by the current class. A separate copy of the instance data is created for each object instance.

*** integer**. (1) A numeric literal that does not include any digit positions to the right of the decimal point.

(2) A numeric data item defined in the DATA DIVISION that does not include any digit positions to the right of the decimal point.

(3) A numeric function whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

integer function. A function whose category is numeric and whose definition does not include any digit positions to the right of the decimal point.

interface. The information that a *client* must know to use a *class*—the names of its *attributes* and the signatures of its *methods*. With direct-to-SOM compilers such as COBOL, the interface to a class may be defined by native language syntax for class definitions. Classes implemented in other languages might have their interfaces defined directly in SOM Interface Definition Language (IDL). The COBOL compiler has a compiler option, IDLGEN, to automatically generate IDL for a COBOL class.

Interface Definition Language (IDL). The formal language (independent of any programming language) by which the *interface* for a class of *objects* is defined in a IDL file, which the SOM compiler then interprets to create an implementation template file and binding files. SOM′s Interface Definition Language is fully compliant with standards established by the Object Management Group′s Common Object Request Broker Architecture (*CORBA*).

interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

intermediate result. An intermediate field containing the results of a succession of arithmetic operations.

*** internal data**. The data described in a program excluding all external data items and external file connectors. Items described in the LINKAGE SECTION of a program are treated as internal data.

*** internal data item**. A data item which is described in one program in a run unit. An internal data item may have a global name.

internal decimal item. A format in which each byte in a field except the rightmost byte represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. (Also known as packed decimal.)

*** internal file connector**. A file connector which is accessible to only one object program in the run unit.

*** intra-record data structure**. The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

intrinsic function. A pre-defined function, such as a commonly used arithmetic function, called by a built-in function reference.

*** invalid key condition**. A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

*** I-O-CONTROL**. The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

*** I-O-CONTROL entry**. An entry in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION which contains clauses that provide information required for the transmission and handling of data on named files during the execution of a program.

*** I-O-Mode**. The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phase for that file.

*** I-O status**. A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

iteration structure. A program processing logic in which a series of statements is repeated while a condition is true or until a condition is true.

K

K. When referring to storage capacity, two to the tenth power; 1024 in decimal notation.

*** key**. A data item that identifies the location of a record, or a set of data items which serve to identify the ordering of data.

*** key of reference**. The key, either prime or alternate, currently being used to access records within an indexed file.

*** key word**. A reserved word or function-name whose presence is required when the format in which the word appears is used in a source program.

kilobyte (KB). One kilobyte equals 1024 bytes.

L

*** language-name**. A system-name that specifies a particular programming language.

Language Environment-conforming. A characteristic of compiler products (COBOL for OS/390 & VM, COBOL for MVS & VM, COBOL/370, AD/Cycle C/370, C/C++ for MVS and VM, PL/I for MVS and VM) that produce object code conforming to the Language Environment format.

last-used state. A program is in last-used state if its internal values remain the same as when the program was exited (are not reset to their initial values).

*** letter**. A character belonging to one of the following two sets:

1. Uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

2. Lowercase letters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

*** level indicator**. Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the DATA DIVISION are: CD, FD, and SD.

*** level-number**. A user-defined word, expressed as a two digit number, which indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range from 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77 and 88 identify special properties of a data description entry.

*** library-name**. A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

*** library text**. A sequence of text words, comment lines, the separator space, or the separator pseudo-text delimiter in a COBOL library.

LILIAN DATE. The number of days since the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582. The Lilian date format is named in honor of Luigi Lilio, the creator of the Gregorian calendar.

*** LINAGE-COUNTER**. A special register whose value points to the current position within the page body.

LINKAGE SECTION. The section in the DATA DIVISION of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

literal. A character-string whose value is specified either by the ordered set of characters comprising the string, or by the use of a figurative constant.

local. A set of attributes for a program execution environment indicating culturally sensitive considerations, such as: character code page, collating sequence, date/time format, monetary value representation, numeric value representation, or language.

*** LOCAL-STORAGE SECTION**. The section of the DATA DIVISION that defines storage that is allocated and freed on a per-invocation basis, depending on the value assigned in their VALUE clauses.

*** logical operator**. One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND, or OR, or both can be used as logical connectives. NOT can be used for logical negation.

*** logical record**. The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group of items. The term is synonymous with record.

*** low order end**. The rightmost character of a string of characters.

M

main program. In a hierarchy of programs and subroutines, the first program to receive control when the programs are run.

*** mass storage**. A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

*** mass storage device**. A device having a large storage capacity; for example, magnetic disk, magnetic drum.

*** mass storage file**. A collection of records that is assigned to a mass storage medium.

*** megabyte (M)**. One megabyte equals 1,048,576 bytes.

*** merge file**. A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

metaclass. A SOM class whose instances are SOM class-objects. The methods defined in metaclasses are executed without requiring any object instances of the class to exist, and are frequently used to create instances of the class.

method. Procedural code that defines one of the operations supported by an object, and that is executed by an INVOKE statement on that object.

*** Method Definition**. The COBOL source unit that defines a method.

*** method identification entry**. An entry in the METHOD-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the method-name and assign selected attributes to the method definition.

*** method-name**. A user-defined word that identifies a method.

*** mnemonic-name**. A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified implementor-name.

multitasking. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks. When running under the Language Environment product, multitasking is synonymous with *multithreading*.

N

name. A word composed of not more than 30 characters that defines a COBOL operand.

*** native character set**. The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** native collating sequence**. The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

*** negated combined condition**. The ‘NOT’ logical operator immediately followed by a parenthesized combined condition.

*** negated simple condition**. The ‘NOT’ logical operator immediately followed by a simple condition.

nested program. A program that is directly contained within another program.

*** next executable sentence**. The next sentence to which control will be transferred after execution of the current statement is complete.

*** next executable statement**. The next statement to which control will be transferred after execution of the current statement is complete.

*** next record**. The record that logically follows the current record of a file.

*** noncontiguous items**. Elementary data items in the WORKING-STORAGE and LINKAGE SECTIONs that bear no hierarchic relationship to other data items.

*** non-numeric item**. A data item whose description permits its content to be composed of any combination of characters taken from the computer’s character set. Certain categories of non-numeric items may be formed from more restricted character sets.

*** non-numeric literal**. A literal bounded by quotation marks. The string of characters may include any character in the computer’s character set.

null. Figurative constant used to assign the value of an invalid address to pointer data items. NULLS can be used wherever NULL can be used.

*** numeric character**. A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

numeric-edited item. A numeric item that is in such a form that it may be used in printed output. It may consist of external decimal digits from 0 through 9, the decimal point, commas, the dollar sign, editing sign control symbols, plus other editing symbols.

*** numeric function**. A function whose class and category are numeric but which for some possible evaluation does not satisfy the requirements of integer functions.

*** numeric item**. A data item whose description restricts its content to a value represented by characters chosen from the digits from ‘0’ through ‘9’; if signed, the item may also contain a ‘+’, ‘-’, or other representation of an operational sign.

*** numeric literal**. A literal composed of one or more numeric characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

O

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

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

*** OBJECT-COMPUTER**. The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the object program is executed, is described.

*** object computer entry**. An entry in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the object program is to be executed.

object deck. A portion of an object program suitable as input to a linkage editor. Synonymous with *object module* and *text deck.*

object module. Synonym for *object deck* or *text deck*.

*** object of entry**. A set of operands and reserved words, within a DATA DIVISION entry of a COBOL program, that immediately follows the subject of the entry.

*** object program**. A set or group of executable machine language instructions and 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. Where there is no danger of ambiguity, the word ‘program’ alone may be used in place of the phrase ‘object program.’

*** object time**. The time at which an object program is executed. The term is synonymous with execution time.

*** obsolete element**. A COBOL language element in Standard COBOL that is to be deleted from the next revision of Standard COBOL.

ODBC. Open Database Connectivity that provides you access to data from a variety of databases and file systems.

ODO object. In the example below,

```
WORKING-STORAGE SECTION
01  TABLE-1.
    05  X                  PICS9.
    05  Y OCCURS 3 TIMES
          DEPENDING ON X    PIC X.
```

X is the object of the OCCURS DEPENDING ON clause (ODO object). The value of the ODO object determines how many of the ODO subject appear in the table.

ODO subject. In the example above, Y is the subject of the OCCURS DEPENDING ON clause (ODO subject). The number of Y ODO subjects that appear in the table depends on the value of X.

*** open mode**. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

*** operand**. Whereas the general definition of operand is “that component which is operated upon,” for the purposes of this document, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

*** operational sign**. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

*** optional file**. A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

*** optional word**. A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

OS/2 (Operating System/2*). A multi-tasking operating system for the IBM Personal Computer family that allows you to run both DOS mode and OS/2 mode programs.

*** output file**. A file that is opened in either the OUTPUT mode or EXTEND mode.

*** output mode**. The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

*** output procedure**. A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

P

packed decimal item. See “internal decimal item.”

*** padding character**. An alphanumeric character used to fill the unused character positions in a physical record.

page. A vertical division of output data representing a physical separation of such data, the separation being based on internal logical requirements and/or external characteristics of the output medium.

*** page body**. That part of the logical page in which lines can be written and/or spaced.

*** paragraph**. In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the IDENTIFICATION and ENVIRONMENT DIVISIONs, a paragraph header followed by zero, one, or more entries.

*** paragraph header**. A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT DIVISIONs. The permissible paragraph headers in the IDENTIFICATION DIVISION are:

```
PROGRAM-ID. (Program IDENTIFICATION DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

The permissible paragraph headers in the ENVIRONMENT DIVISION are:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

*** paragraph-name**. A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. Parameters are used to pass data values between calling and called programs.

password. A unique string of characters that a program, computer operator, or user must supply to meet security requirements before gaining access to data.

*** phrase**. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

*** physical record**. See "block."

pointer data item. A data item in which address values can be stored. Data items are explicitly defined as pointers with the USAGE IS POINTER clause. ADDRESS OF special registers are implicitly defined as pointer data items. Pointer data items can be compared for equality or moved to other pointer data items.

portability. The ability to transfer an application program from one application platform to another with relatively few changes to the source program.

preloaded. In COBOL this refers to COBOL programs that remain resident in storage under IMS instead of being loaded each time they are called.

*** prime record key**. A key whose contents uniquely identify a record within an indexed file.

*** priority-number**. A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters ′0′,′1′, ... , ′9′. A segment-number may be expressed either as a one- or two-digit number.

*** procedure**. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

*** procedure branching statement**. A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written in the source program. The procedure branching statements are: ALTER, CALL, EXIT, EXIT PROGRAM, GO TO, MERGE, (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

Procedure Division. One of the four main component parts of a COBOL program, class definition, or method definition. The Procedure Division contains instructions for solving a problem. The Program and Method Procedure Divisions may contain imperative statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections. The Class Procedure Division contains only method definitions.

procedure integration. One of the functions of the COBOL optimizer is to simplify calls to performed procedures or contained programs.

PERFORM procedure integration is the process whereby a PERFORM statement is replaced by its performed procedures. Contained program procedure integration is the process where a CALL to a contained program is replaced by the program code.

*** procedure-name**. A user-defined word that is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified) or a section-name.

procedure-pointer data item. A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS PROCEDURE-POINTER clause contains the address of a procedure entry point.

*** program identification entry**. An entry in the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the program-name and assign selected program attributes to the program.

*** program-name**. In the IDENTIFICATION DIVISION and the end program header, a user-defined word that identifies a COBOL source program.

*** pseudo-text**. A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

*** pseudo-text delimiter**. Two contiguous equal sign characters (==) used to delimit pseudo-text.

*** punctuation character**. A character that belongs to the following set:

Character	**Meaning**
,	comma
;	semicolon
:	colon
.	period (full stop)
″	quotation mark
(	left parenthesis
)	right parenthesis
•	space
=	equal sign

Q

QSAM (Queued Sequential Access Method). An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

*** qualified data-name**. An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

*** qualifier**.

1. A data-name or a name associated with a level indicator which is used in a reference either together with another data-name which is the name of an item that is subordinate to the qualifier or together with a condition-name.
2. A section-name that is used in a reference together with a paragraph-name specified in that section.
3. A library-name that is used in a reference together with a text-name associated with that library.

R

*** random access**. An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

*** record**. See “logical record.”

*** record area**. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section of the DATA DIVISION. In the File Section, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

*** record description**. See “record description entry.”

*** record description entry**. The total set of data description entries associated with a particular record. The term is synonymous with record description.

recording mode. The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), S (spanned), or U (undefined).

record key. A key whose contents identify a record within an indexed file.

*** record-name**. A user-defined word that names a record described in a record description entry in the DATA DIVISION of a COBOL program.

*** record number**. The ordinal number of a record in the file whose organization is sequential.

recursion. A program calling itself or being directly or indirectly called by a one of its called programs.

recursively capable. A program is recursively capable (can be called recursively) if the RECURSIVE attribute is on the PROGRAM-ID statement.

reel. A discrete portion of a storage medium, the dimensions of which are determined by each implementor that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

reentrant. The attribute of a program or routine that allows more than one user to share a single copy of a load module.

*** reference format**. A format that provides a standard method for describing COBOL source programs.

reference modification. A method of defining a new alphanumeric data item by specifying the leftmost character and length relative to the leftmost character of another alphanumeric data item.

*** reference-modifier**. A syntactically correct combination of character-strings and separators that defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

*** relation**. See “relational operator” or “relation condition.”

*** relational operator**. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Operator	**Meaning**
IS GREATER THAN	Greater than
IS >	Greater than
IS NOT GREATER THAN	Not greater than
IS NOT >	Not greater than
IS LESS THAN	Less than
IS <	Less than
IS NOT LESS THAN	Not less than
IS NOT <	Not less than
IS EQUAL TO	Equal to
IS =	Equal to
IS NOT EQUAL TO	Not equal to
IS NOT =	Not equal to

IS GREATER THAN OR EQUAL TO	Greater than or equal to
IS >=	Greater than or equal to
IS LESS THAN OR EQUAL TO	Less than or equal to
IS <=	Less than or equal to

*** relation character**. A character that belongs to the following set:

Character	**Meaning**
>	greater than
<	less than
=	equal to

*** relation condition**. The proposition, for which a truth value can be determined, that the value of an arithmetic expression, data item, non-numeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, non-numeric literal, or index name. (See also "relational operator.")

*** relative file**. A file with relative organization.

*** relative key**. A key whose contents identify a logical record in a relative file.

*** relative organization**. The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

*** relative record number**. The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

*** reserved word**. A COBOL word specified in the list of words that may be used in a COBOL source program, but that must not appear in the program as user-defined words or system-names.

*** resource**. A facility or service, controlled by the operating system, that can be used by an executing program.

*** resultant identifier**. A user-defined data item that is to contain the result of an arithmetic operation.

reusable environment. A reusable environment is when you establish an assembler program as the main program by using either ILBOSTP0 programs, IGZERRE programs, or the RTEREUS run-time option.

routine. A set of statements in a COBOL program that causes the computer to perform an operation or series of related operations. In Language Environment, refers to either a procedure, function, or subroutine.

*** routine-name**. A user-defined word that identifies a procedure written in a language other than COBOL.

*** run time**. The time at which an object program is executed. The term is synonymous with object time.

run-time environment. The environment in which a COBOL program executes.

*** run unit**. A stand-alone object program, or several object programs, that interact via COBOL CALL statements, which function at run time as an entity.

S

SBCS (Single Byte Character Set). See "Single Byte Character Set (SBCS)."

scope terminator. A COBOL reserved word that marks the end of certain Procedure Division statements. It may be either explicit (END-ADD, for example) or implicit (separator period).

*** section**. A set of zero, one or more paragraphs or entities, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

*** section header**. A combination of words followed by a separator period that indicates the beginning of a section in the Environment, Data, and Procedure Divisions. In the ENVIRONMENT and DATA DIVISIONs, a section header is composed of reserved words followed by a separator period. The permissible section headers in the ENVIRONMENT DIVISION are:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

The permissible section headers in the DATA DIVISION are:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a separator period.

*** section-name**. A user-defined word that names a section in the Procedure Division.

selection structure. A program processing logic in which one or another series of statements is executed, depending on whether a condition is true or false.

*** sentence**. A sequence of one or more statements, the last of which is terminated by a separator period.

*** separately compiled program**. A program which, together with its contained programs, is compiled separately from all other programs.

*** separator**. A character or two contiguous characters used to delimit character-strings.

*** separator comma**. A comma (,) followed by a space used to delimit character-strings.

*** separator period**. A period (.) followed by a space used to delimit character-strings.

*** separator semicolon**. A semicolon (;) followed by a space used to delimit character-strings.

sequence structure. A program processing logic in which a series of statements is executed in sequential order.

*** sequential access**. An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

*** sequential file**. A file with sequential organization.

*** sequential organization**. The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

serial search. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last.

*** 77-level-description-entry**. A data description entry that describes a noncontiguous data item with the level-number 77.

*** sign condition**. The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

*** simple condition**. Any single condition chosen from the set:

Relation condition
Class condition
Condition-name condition
Switch-status condition
Sign condition

Single Byte Character Set (SBCS). A set of characters in which each character is represented by a single byte. See also ″EBCDIC (Extended Binary-Coded Decimal Interchange Code).″

slack bytes. Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

SOM. See “System Object Model”

*** sort file**. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

*** sort-merge file description entry**. An entry in the File Section of the DATA DIVISION that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

*** SOURCE-COMPUTER**. The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the source program is compiled, is described.

*** source computer entry**. An entry in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the source program is to be compiled.

*** source item**. An identifier designated by a SOURCE clause that provides the value of a printable item.

source program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with the IDENTIFICATION DIVISION or a COPY statement. A COBOL source program is terminated by the end program header, if specified, or by the absence of additional source program lines.

*** special character**. A character that belongs to the following set:

Character	**Meaning**
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	slant (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
″	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

*** special-character word**. A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an ENVIRONMENT DIVISION paragraph in which environment-names are related to user-specified mnemonic-names.

*** special names entry**. An entry in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating sequences; and relating class-names to sets of characters.

*** special registers**. Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

*** standard data format**. The concept used in describing the characteristics of data in a COBOL DATA DIVISION under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

*** statement**. A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

STL. Standard Language file system: native workstation and PC file system for COBOL and PL/I. Supports sequential, relative, and indexed files, including the full ANSI 85 COBOL standard I/O language and all of the extensions described in *&lrcit..IBM COBOL Language Reference*, unless exceptions are explicitly noted.

structured programming. A technique for organizing and coding a computer program in which the program comprises a hierarchy of segments, each segment having a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the hierarchy.

*** sub-class**. A class that inherits from another class. When two classes in an inheritance relationship are considered together, the sub-class is the inheritor or inheriting class; the *super-class* is the inheritee or inherited class.

*** subject of entry**. An operand or reserved word that appears immediately following the level indicator or the level-number in a DATA DIVISION entry.

*** subprogram**. See "called program."

*** subscript**. An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, that identifies a particular element in a table. A subscript may be the word ALL when the subscripted identifier is used as a function argument for a function allowing a variable number of arguments.

*** subscripted data-name**. An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

*** super-class**. A class that is inherited by another class. See also *sub-class*.

switch-status condition. The proposition, for which a truth value can be determined, that an UPSI switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

*** symbolic-character**. A user-defined word that specifies a user-defined figurative constant.

syntax. (1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (2) The structure of expressions in a language. (3) The rules governing the structure of a language. (4) The relationship among symbols. (5) The rules for the construction of a statement.

*** system-name**. A COBOL word that is used to communicate with the operating environment.

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

T

*** table**. A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

*** table element**. A data item that belongs to the set of repeated items comprising a table.

text deck. Synonym for *object deck* or *object module*.

*** text-name**. A user-defined word that identifies library text.

*** text word**. A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

- A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for non-numeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, source program, or pseudo-text, are always considered text words.

- A literal including, in the case of non-numeric literals, the opening quotation mark and the closing quotation mark that bound the literal.
- Any other sequence of contiguous COBOL characters except comment lines and the word ‘COPY’ bounded by separators that are neither a separator nor a literal.

top-down design. The design of a computer program using a hierarchic structure in which related functions are performed at each level of the structure.

top-down development. See “structured programming.”

trailer-label. (1) A file or data set label that follows the data records on a unit of recording medium. (2) Synonym for end-of-file label.

*** truth value**. The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

U

*** unary operator**. A plus (+) or a minus (-) sign, that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

unit. A module of direct access, the dimensions of which are determined by IBM.

universal object reference. A data-name that can refer to an object of any class.

*** unsuccessful execution**. The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

UPSI switch. A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

*** user-defined word**. A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

V

*** variable**. A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

*** variable length record**. A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

*** variable occurrence data item**. A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

*** variably located group.**. A group item following, and not subordinate to, a variable-length table in the same level-01 record.

*** variably located item.**. A data item following, and not subordinate to, a variable-length table in the same level-01 record.

*** verb**. A word that expresses an action to be taken by a COBOL compiler or object program.

VM/SP (Virtual Machine/System Product). An IBM-licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a “real” machine.

volume. A module of external storage. For tape devices it is a reel; for direct-access devices it is a unit.

volume switch procedures. System specific procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

W

*** word**. A character-string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word, or a function-name.

*** WORKING-STORAGE SECTION**. The section of the DATA DIVISION that describes working storage data items, composed either of noncontiguous items or working storage records or of both.

Z

zoned decimal item. See “external decimal item.”

List of Abbreviations

APAR	authorized program analysis report
API	application program interface
APPC	advanced program-to-program communication
APPN	advanced peer-to-peer network
ASCII	American National Standard Code for Information Interchange
C&SM	communication and systems management
CICS	customer information control system
CICS/DDM	CICS.distributed data management
COBOL	Common Business Oriented Language
CSD	corrective service diskette
DB2	DATABASE 2
DBCS	double byte character set
DFM	distributed file manager
DFSMS	data facility storage management
DFSMShsm	DFSMS hierarchical storage manager
DFSMS/MVS	DFSMS/multiple virtual storage
DLC	data link control
EBCDIC	extended binary coded decimal interchange code
FAT	file allocation table
FTP	file transfer program
GID	group identification
GUI	graphical user interface
HPFS	high performance file system
I/O	input/output
IBM	International Business Machines Corporation
IEEE	Institute of Electrical and Electronic Engineers
IP	Internet protocol
ISPF	interactive system productivity facility
ITSO	International Technical Support Organization
JCL	job control language
LAN	local area network
LAPS	lan adapter and protocol support
LPEX	live parsing editor
LU	logical unit
MPTN	multi-protocol transport networking
MVS	multiple virtual storage
MVS/ESA	MVS/enterprise systems architecture
NFS	network file system
NFSCTL	network file system client latest
NT	new technology (Microsoft Windows NT)
OS/2	operating system/2
OS/390	operating system/390
OS/400	operating system/400
PC	personal computer
PCNFSD	personal computer network file system daemon
PDS	partitioned data set
PDSE	partitioned data set extended
PING	packet internet groper
REXX	restructured extended executor language
RPC	remote procedure call
SAM	sequential access method
SdU	SMARTdata Utilities
SNA	Systems Network Architecture
SQL	structured query language
TCP/IP	transmission control protocol/internet protocol
TP	transaction program
TSO	time sharing option
UDP	user datagram protocol
URL	uniform resource locator
VM	virtual machine
VSAM	virtual storage access method

XID	exchange identifier

Index

A

abbreviations 131
acronyms 131
APPC
  Communications Manager/2 installation 31
  configuration 30
  host configuration 54
  product requirement 3
  SMARTdata Utilities 53
  workstation connection to host 51

B

bibliography 105

C

Communications Manager/2
  APPC installation 31
  configuring for APPC 40
compiling
  host application 73

D

DB2
  host database creation 85
  sample application 84
debugger
  application 76
  breakpoints 84
  call stack 83
  debugger 80
  step into 80
  step over 79
drive mapping
  specifications 13

E

Editing
  through WorkFrame 69

F

file name mapping
  definition in MVSINFO 13
  host configuration 21
  support for 16

G

glossary 111

H

host access
  starting from workstation 17

J

JCL members
  creation 89
job status
  through WorkFrame 73

M

MVS data types
  sample 93
MVSINFO file
  overview 12

N

NFS kit
  APAR installation 11
  APAR requirement 4
  CSD installation 11
  CSD requirement 4
  software requirement 4

O

OS/2
  APPC configuration 29
  TCP/IP configuration 4
  version required 3

P

protected saving
  overview 17

R

remote compiling
  through WorkFrame 73
remote debugging
  through WorkFrame 76
Remote editing
  through WorkFrame 69

S

SMARTdata Utilities
  configuration with APPC 53
  overview 29
submitting MVS jobs
  through WorkFrame 73

T

TCP/IP
  configuring 4
  host connection test 11
  IP address 5
  mount command 12
  NFS kit requirement 4
  NFS mapping 24
  PING command 6
  product requirement 3
  using FTP 25

U

userid
  in MVSINFO.DAT 13

V

VSAM
  accessing data 93
  creating host file 94
  relation to SMARTdata Utilities 94
  software requirements 3

W

WorkFrame
  defining an application 59
  MVS edit 69
  MVS job status 73, 74
  MVS SYSOUT 73
  submit MVS job 73

ITSO Redbook Evaluation

VisualAge 2000 - Remote Edit, Compile, and Debug Using VisualAge COBOL 2.0 on OS/2
SG24-2245-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____________

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:
____

____

____

____

What other redbooks would you like to see published?
____

____

____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**
____

____

____

____

____

Printed in U.S.A.

SG24-2245-00