IBM

NetView/PC™

SC30-3313-1

Application Program Interface/ Communications Services

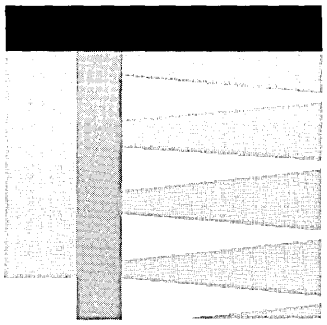
Version 1.1

NetView/PC™

SC30-3313-1

Application Program Interface/ Communications Services

Version 1.1



File Number
S370/4300/30XX-50

Program Number
5669-024

Second Edition (October 1987)

This edition is a major revision of SC30-3313-0. It applies to NetView/PC™, program number 5669-024, and to all subsequent releases and modifications until otherwise indicated in new editions.

Changes are made periodically to the information herein; before you use this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Contents

Part 1. Application Program Interface/Communications Services Overview 1

Chapter 1. Network Management Overview 3
What is the Environment 3
NetView/PC in Communications Network Management 5

Chapter 2. Application Program Interface/Communications Services (API/CS) . 9
Using the API/CS 10
API/CS Scenario 13
Programming in the NetView/PC DOS Partition 16
Requirements and Restrictions 17
Installation 18
Using EZ−VU 18
Verified PC DOS Applications 19
Applications Not Successfully Executed 19
Relating to Documentation 20

Part 2. Using the API/CS Subroutines 23

Chapter 3. Alert Subroutine Calls 25
Open the Alert API/CS 30
Send an Alert 30
Close the Alert API/CS 30

Chapter 4. Operator Communications Subroutine Calls 31
Open the Operator Communications API/CS 32
Write the Icon 'DP' to the NetView/PC Icon Window 32
Clear the Icon from the NetView/PC Icon Window 33
Close the Operator Communications API/CS 33

Chapter 5. Service Point Command Facility (SPCF) Subroutine Calls 35
Open the SPCF API/CS 40
Receive a RUNCMD message 40
Send a RUNCMD response 41
Send a Message 42
Receive a Command 43
Send a Command Response 43
Send Error Sense 44
Close the SPCF API/CS 45

Chapter 6. SPCF Build and Parse 47
Parse 47
Build 51

Chapter 7. Host Data Facility Subroutine Calls 57
Open the Host Data Facility API/CS 60
Send File Data 60
Receive File Data 61
Check the Status of a Host Data Facility Request 61

Stop File Data Transfer 61
Close the Host Data Facility API/CS 62

Part 3. Reference Information 63

Appendix A. API/CS Reference Information 67
Return Code List 67
DOS Error Codes 71
Translation of NMVT Data Fields 72
Naming Conventions 73

Appendix B. Alert Major Vector Formats 77
Non-generic Alert Format 77
Tables of Text for X'91' Subvector Support 78
NetView/PC X'9F' Subvector 80
NetView/PC ALERT SV X'9F' Code Point File : DUPALGTF.TXT 83
Generic Alert Format 87

Appendix C. Service Point Command Data 89
API/CS Supported NetView Commands 89
Service Point Command vectors 94

Appendix D. Suggested Command Formats 107
Suggested Physical Device Management Commands 107
Configuration Data Base Management Commands 108

Appendix E. Panel Development Rules 113
Applicability and Conformance 113
Panel Design 113
Panel Dialog Management 137
File Management Techniques Using the List Panel 155

Part 4. Sample Programs 163

Appendix F. DOS Sample Program Planning and Installation 167
Prerequisites 167
Components 167
Installation 170
Assembly and Link 170

Appendix G. Operation 173
Startup 173
General Information 173
Main Menu 176
Alert Interface Panel 177
Operator Communications Interface 179
Service Point Command Facility Interface 181
SPCF Run Command 182
SPCF Receive Unparsed Command 185
SPCF Parse Host Command 187
SPCF Build Response 189
SPCF Display File 190
SPCF Unformatted Display Panel 192
SPCF Display Run Command 194

SPCF Display Link PD Command 195
SPCF Display Link Data Command 197
SPCF Display Link Test Command 198
SPCF Send Unformatted Response 200
SPCF Send Unsolicited Message 202
SPCF Send Error Sense Data 204
SPCF Build Message 207
SPCF Correlator List 208
Host Data Facility Interface 209

Appendix H. API Sample Program Error Messages 213

Appendix I. DOS Sample Program Code 223
APIMAIN.DSG 223
APIMAIN.MAC 242
APIMAIN.EXR 245
APIMAIN.DEF 246
APIMAIN.ASM 247
APIMAIN.UTL 292
APIUTIL.DSG 303
APIUTIL.EXR 310
APIUTIL.ASM 311
APIDISP.DSG 332
APIDISP.ASM 340

Appendix J. NetView Sample Programs 361
NetView Sample Presentation Services Command Processor (PSCP) 361
NetView Sample Data Services Command Processor (DSCP) 371

Glossary 389

Index 407

Figures

1. Open Network Management Concept . . . 3
2. Network Management Structure . . . 5
3. Network Management Products . . . 6
4. Network Management and Vendor Products . . . 6
5. API/CS Network Management Scenario. . . . 14
6. Documentation for the Environment. . . . 21
7. Alert ARB . . . 26
8. Primary Alert API/CS Return Codes . . . 27
9. Secondary Alert API/CS Return Codes . . . 27
10. Operator Communications ARB . . . 31
11. Operator Communications API/CS Return Codes . . . 32
12. SPCF ARB . . . 37
13. SPCF API/CS Return Codes . . . 39
14. Message buffer format when Convert = 'Y' . . . 42
15. SENSETYPE Values, Data and Descriptions . . . 44
16. Defined LCCSTAT Values . . . 45
17. Defined Error Detail Values . . . 45
18. Parse SPCF Command ARB . . . 48
19. Parse SPCF Command API/CS Return Codes . . . 49
20. Parse Data fields and ARB Displacements Returned . . . 50
21. PARSE Sense Data and Description . . . 50
22. Format for Names List . . . 51
23. Build SPCF Reply ARB . . . 51
24. Build SPCF Reply API/CS Return Codes . . . 52
25. Path Configuration Information CB . . . 54
26. LINKPD LCC Description CB . . . 54
27. LINKDATA and LINKTEST LCC Description CB . . . 55
28. LINKDATA and LINKTEST LCC Data CB . . . 55
29. Format for Probable Cause Data . . . 56
30. Host Data Facility ARB . . . 57
31. Host Data Facility API/CS Return Codes . . . 58
32. List of all API/CS Return Codes . . . 67
33. ASCII to EBCDIC Translation . . . 73
34. EBCDIC to ASCII Translation . . . 73
35. NetView/PC Non-generic Alert NMVT Example . . . 78
36. SV X'91' Alert Type Field, File : DUPALATF.TXT . . . 79
37. SV X'91' Cause Code Field, File : DUPALGCF.TXT . . . 79
38. SV X'91' Specific Component Code Field, File : DUPALSCF.TXT . . . 80
39. NetView/PC Alert . . . 80
40. NetView/PC X'9F' Subvector fields. . . . 80
41. NetView/PC X'9F' Subfields. . . . 81
42. Probable Cause subfield of NetView/PC Alert Data Subvector . . . 82
43. User Cause Subfield of the NetView/PC Alert Data Subvector . . . 82
44. Install Cause Subfield of the NetView/PC Alert Data Subvector . . . 83
45. Failure Cause Subfield . . . 83
46. X'9F' subvector Alert Description records . . . 84
47. Probable Cause records . . . 84
48. User Cause records . . . 85
49. Install Cause records . . . 86
50. Failure Cause records . . . 86
51. Recommended Action records . . . 86
52. NetView/PC Generic Alert NMVT Example . . . 88

53. Max NMVT length possible for the LINKPD "algorithm" 94
54. Max NMVT length possible for the LINKDATA "algorithm" 94
55. Max NMVT length possible for the PUT LINKTEST "algorithm" 95
56. NMVT Header 95
57. RUNCMD 95
58. Sense Reply to RUNCMD 96
59. Formatted Response message to RUNCMD 96
60. Unformatted Response message to RUNCMD 96
61. Send Message To Operator 96
62. LINKPD 97
63. Sense Response to LINKPD 97
64. Response to LINKPD 97
65. Link Status Subvector 97
66. Probable Cause Subvector 98
67. LINKDATA 98
68. Sense Response to LINKDATA 98
69. Response to LINKDATA 98
70. LCC data subvector 99
71. LINKTEST 99
72. Test Set Up Data Subvector 99
73. Sense Response to LINKTEST 100
74. Response to LINKTEST 100
75. Link Test Results Subvector 101
76. LCC data subvector 102
77. Hierarchy/Resource List Subvector 102
78. Name List Subvector 103
79. Qualified Message Subvector 104
80. Text Message Subvector 104
81. Reply Count Subvector 105
82. Sense Data Subvector 105
83. Common Panel Elements 115
84. Common Panel Elements 119
85. Panel Body Elements 120
86. Panel Body Elements 126
87. Panel Body Element Headings 126
88. Single Selection Field Menu Panel - Panel Body 127
89. Single Selection Field Menu Panel - Example 1 128
90. Single Selection Field Menu Panel - Example 2 128
91. Multiple Selection Field Menu Panel - Panel Body 128
92. Multiple Selection Field Menu Panel - Example 1 129
93. Parameter Entry Panel Body Elements 130
94. Parameter Entry Panel - Example 1 131
95. Vertical Data Entry Panel Format 132
96. Tabular Data Entry Panel Format 132
97. Forms Fill-in Data Entry Panel Format 133
98. List Panel Format 133
99. List Panel - Example 1 134
100. Information Panel Example 135
101. Mixed Panel Example: Entry and Menu 136
102. Mixed Panel Example: Information, Entry and Menu 136
103. Portion of a Possible Panel Hierarchy (4 levels) 142
104. NetView/PC EZ-VU II Level of Emphasis 152
105. Class of Data Versus Level of Emphasis Number 153
106. Level of Emphasis Assignment for Output Data Classes 154
107. Level of Emphasis Assignment for Input Data Classes 155
108. Level of Emphasis Assignment for Message Sub-classes 155

109. Level of Emphasis Assignment for Status Text Sub-classes 155
110. Panel Hierarchy Example 157
111. Main Menu 176
112. Alert Interface Panel 177
113. Operator Communications Interface Panel 179
114. SPCF Main Menu 181
115. SPCF Run Command Panel 182
116. SPCF Get No Parse Panel 185
117. SPCF Parse Command Panel 187
118. SPCF Build Response Panel 189
119. SPCF Display File Pop-Up Panel 190
120. SPCF Display Unformatted Format Panel 192
121. SPCF Display Formatted Run Command Panel 194
122. SPCF Display Formatted Link PD Command Panel 195
123. SPCF Display Formatted Link Data Command Panel 197
124. SPCF Display Formatted Link Test Command Panel 198
125. SPCF Put Unformatted Panel 200
126. SPCF Send Message Panel 202
127. SPCF Send Error Panel 204
128. SPCF Message Buffer Panel 207
129. SPCF Correlator Selection Panel 208
130. Host Data Facility Interface Panel 209
131. Example of a Language Statement 392
132. NCP Example 392
133. VTAM Examples 393
134. Links and Path Controls 396

About This Book

This book describes how to use the NetView/PC Application Program Interface/Communications Services (API/CS). The API/CS is the open network management API in NetView/PC. It describes how to write network management application code that will use the API/CS in order to participate with IBM's centralized network management environment. It describes how to use the NetView/PC API/CS to allow non-SNA network management components to be managed from and by NetView[1] through a vendor or user-supplied network management applicatio

It will guide the PC programmer in the use of NetView/PC API/CS calls and provide the information necessary to build the data structures required to use the API/CS. Reference material is contained in the appendices.

It will not identify other equipment manufacturers (OEM) network component alert conditions or define interfaces, protocols, or procedures to be used between the network management application and the managed components.

This book is organized into four parts:

Part 1, "Application Program Interface/Communications Services Overview," provides a high-level network management overview and describes what you must do to use the API/CS.

- Chapter 1, "Network Management Overview," provides a high-level overview of the network management and explains where your network management application fits in the network management structure.
- Chapter 2, "Application Program Interface/Communications Services (API/CS)," describes the API/CS interface and the communications services provided. It contains a suggested flow of API/CS usage, a scenario describing how the API/CS might be used by a network management application, and a list of the requirements and restrictions imposed on you in order for your application to execute in the DOS partition of NetView/PC.

  It describes how the API/CS subroutines are installed, and the steps your application should code to use the API/CS. It also contains information about using the *EZ–VU II Development Facility for the IBM PC* (EZ–VU), and documentation relating to the environment in which you network management application will execute.

Part 2, "Using the API/CS Subroutines," describes how to build the data structures necessary to make calls to the API/CS.

- Chapter 3, "Alert Subroutine Calls," describes how to use the alert interface to send alerts.
- Chapter 4, "Operator Communications Subroutine Calls," describes how the application can use the API/CS to get the attention of the NetView/PC operator.
- Chapter 5, "Service Point Command Facility (SPCF) Subroutine Calls," describes how to receive and respond to commands from NetView. The API/CS

[1] NetView is a registered trade mark of IBM Corporation.

has been enhanced to provide additional Service Point Commands and to send an unsolicited message to a NetView operator.

- Chapter 6, "SPCF Build and Parse," describes how to use the new build and parse subroutines to parse a received Network Manage Vector Transport (NMVT) and to build a response NMVT.
- Chapter 7, "Host Data Facility Subroutine Calls," describes how to control the transfer of file data to or from the host.

Part 3, "Reference Information," contains reference information.

- Appendix A, "API/CS Reference Information," contains a complete list of the API/CS return codes, the DOS error codes returned to the application, the translate table used to translate NMVT EBCDIC fields, and naming conventions used by NetView/PC.
- Appendix B, "Alert Major Vector Formats," contains information about alert NMVTs that is unique to the NetView/PC environment.
- Appendix C, "Service Point Command Data," describes the supported NetView commands and the NMVTs used for those commands.
- Appendix D, "Suggested Command Formats," suggests command formats for physical device and configuration data base commands.
- Appendix E, "Panel Development Rules," suggests rules for consistent user display interface.

Part 4, "Sample Programs," contains sample program information.

- Appendix F, "DOS Sample Program Planning and Installation," describes the DOS sample program planning and installation.
- Appendix G, "Operation," tells you how to operate the DOS sample program to exercise the API/CS.
- Appendix H, "API Sample Program Error Messages," contains the DOS sample program messages.
- Appendix I, "DOS Sample Program Code," contains the DOS sample program source code. The code is also contained on the diskettes included with this book.
- Appendix J, "NetView Sample Programs," contains the NetView command processor source code. The code is also contained on the diskettes included with this book.

Who Should Use This Book

This book is for IBM Personal Computer (PC) programmers responsible for writing network management applications that will run in the DOS partition of NetView/PC and will use the API/CS. The PC application programmer should be experienced with the IBM PC Macro Assembler 2.0, the IBM PC hardware, Disk Operation System (DOS) 3.3, and familiar with NetView/PC.

It is also for system programmers who will write or modify NetView command processors to handle unformatted RUNCMD response messages. An understanding of NetView is also required to use the Service Point Command Facility (SPCF) of NetView/PC.

How to Use this Book

You should be familiar with network management concepts and the IBM network management products before you try to design and write a network management application intended to use the NetView/PC API/CS. You should read through Chapter 1, "Network Management Overview," and Chapter 2, "Application Program Interface/Communications Services (API/CS)," before turning to the appropriate section for each function that is to be used.

You should read and understand "Naming Conventions," and Appendix E, "Panel Development Rules," before you design and develop panels and before you write any DOS applications.

If you have any questions about installation or use of your PC, refer to the version of the *IBM Guide to Operations* for your PC, or the IBM Disk Operation System (DOS) documentation.

What is New and Changed?

New and changed items in this book are:

- Information retrievability has been improved by restructuring the book into four parts, providing more heading to help find information, and providing page numbers when referencing figures or subjects in this book.
- A high-level network management overview was added to Chapter 1, "Network Management Overview," to show where the vendor written NetView/PC application using the API/CS fit in the network management structure.
- Chapter 5, "Service Point Command Facility (SPCF) Subroutine Calls," was changed to reflect support for new commands.
- Chapter 6, "SPCF Build and Parse," is new. This section describes utility routines that help parse received commands and help build responses (NMVTS) to those commands.
- Appendix B, "Alert Major Vector Formats," has descriptions of generic alert and hybrid alert NMVTS, as well as the non-generic alert NMVTS described in the previous level.
- Appendix F, "DOS Sample Program Planning and Installation," "Installation," and Appendix G, "Operation," are new.
- A glossary has been added.
- Sample program diskettes are now provided with this book. The diskettes contain the assembler language source code for the DOS sample program and for the NetView^{TM} command processors. Each sample program on the diskettes, and the listings in this book have the following comments:

  API Sample Program – (C) Copyright IBM Corp. 1986, 1987
  SAMPLE PROGRAM – NO WARRANTY EXPRESSED OR IMPLIED

  You are hereby licensed to use, reproduce, and distribute these sample programs as your needs require. IBM does not warrant the suitability or integrity of these sample programs and accepts no responsibility for their use for your applications. If you choose to copy and redistribute significant portions of these sample programs, you should preface such copies with this copyright notice.

Where to Find More Information

The following lists contain the names and order numbers of documents relating to IBM products and architectures relating to NetView/PC and user-supplied applications executing in the NetView/PC partition in memory. Documents cited in the text of this manual will not have the order numbers with the citation.

NetView/PC Documentation

Information about NetView/PC is found in the following:

- *IBM NetView/PC Planning and Operation Guide*, SC30-3408
- *IBM NetView/PC Version 1.1 API/CS* (API/CS), SC30-3313
- *IBM NetView/PC Version 1.1 Installation Guide*, SC30-3482

Related NetView Documentation

Information about NetView is found in the following:

- Help facility:

  NetView on-line information provides on-line NetView help desk information for NetView operators. It consists of the following major parts:

  - Index
  - Session monitor/hardware monitor glossaries
  - Commands
  - Component overviews
  - VTAM
  - Help desk
  - Recommended actions

- *Learning About NetView: Network Concepts*, SK2T-0292: The PC-based NetView tutorial is an on-line teaching tool. It uses graphics, animation, and NetView screen simulation to introduce new NetView users to network management using NetView.

- *NetView Installation and Administration Guide*, SC30-3360

- *NetView Administration Reference*, SC30-3361

- *NetView Command Lists*, SC30-3423

- *NetView Command Summary*, SX27-3620

- *NetView Customization*, SC30-3462

- *NetView Diagnosis*, LY30-5587

- *NetView Hardware Problem Determination Reference*, SC30-3366

- *NetView Installation and Administration Guide*, SC30-3360,

- *NetView Licensed Program Specifications*, GC30-9589, MVS/VM

- *NetView Messages*, SC30-3365

- *NetView Operation*, SC30-3364

- *NetView Operation Primer*, SC30-3363

- *NetView Operation Scenarios*, SC30-3376
- *Network Program Products General Information*, GC30-3350
- *Network Program Products Bibliography and Master Index*, SC30-3353
- *Network Program Products Planning*, SC30-3351
- *Network Program Products Samples*, SC30-3352
- *Network Program Products Storage Estimates*, SC30-3403

Other Related Documentation

- *IBM Customer Information Control System/VS (CICS/VS)/Distributed Data Management Target Users Guide*, SC21-8066
- Network Control Program and System Support Programs Resource Definition Guide (abbreviated title — *Resource Definition Guide*), SC30-3349
- Network Control Program and System Support Programs Resource Definition Reference (abbreviated title — *Resource Definition Reference*), SC30-3254
- *Disk Operating System 3.3*, 6280060
- *EZ—VU II Development Facility for the IBM PC*, 6410980
- *Systems Network Architecture Formats*, GA27-3136
- *IBM PC Macro Assembler 2.0*, 6024193
- *System Network Architecture Formats*, GA27-3136

Part 1. Application Program Interface/Communications Services Overview

Chapter 1. Network Management Overview 3
What is the Environment 3
  Network Management Structure 4
NetView/PC in Communications Network Management 5
  Where Your Network Management Application Fits 7

Chapter 2. Application Program Interface/Communications Services (API/CS) . 9
Using the API/CS 10
API/CS Scenario 13
Programming in the NetView/PC DOS Partition 16
Requirements and Restrictions 17
Installation 18
Using EZ–VU 18
  EZ–VU Calls 18
  Directories 19
Verified PC DOS Applications 19
Applications Not Successfully Executed 19
Relating to Documentation 20

Chapter 1. Network Management Overview

The purpose of this chapter is to show you where your network management application and the network component being managed fit into IBM'S Open Network Management.

This chapter describes, at a very high level, the environment in which your NetView/PC network management application will execute and the concepts of Open Network Management. It describes IBM'S network management structure, shows where NetView/PC fits in the Open Network Management Architecture, and shows where your application fits into the network management structure.

This chapter also describes the network management services available to your network management application when using the NetView/PC API/CS.

Following chapters describe how to use the API/CS.

What is the Environment

Today's information network is built from diverse technologies. It consists of many components, both hardware and software, and carries multiple information forms. IBM has enhanced the openness of its communication and related architectures by providing new support and new network management capabilities. These architectures are open to enable the attachment of communication products to SNA network management.

The open architectures define the facilities and processes necessary to efficiently connect and manage SNA, non-SNA, IBM and OEM information network components. The concept of three network management product roles, and their relationships to each other, is illustrated in Figure 1.

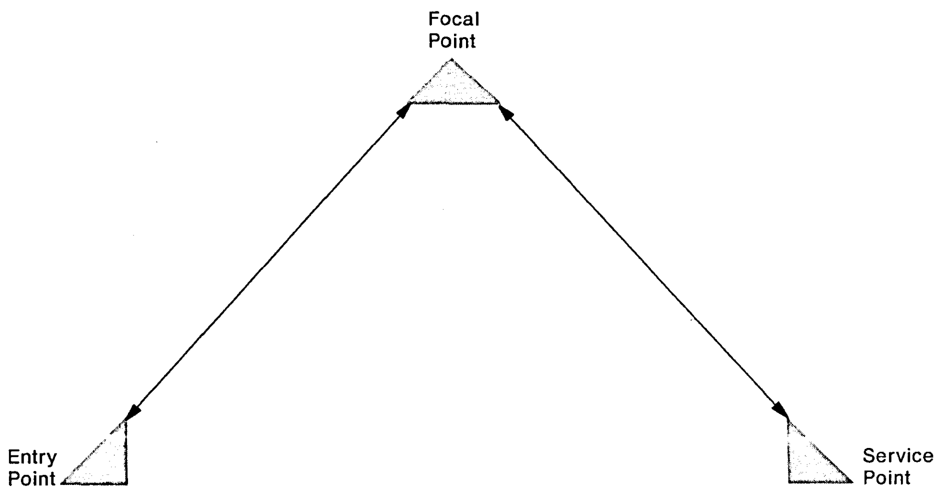


Figure 1. Open Network Management Concept

Introduction to IBM's Open Network Management, SC30-3431 contains the following definitions for the focal point, entry point, and service point.

Focal Point: A network management focal point is a product or set of products that provides centralized network management support. The focal point manages all of the remotely and locally attached network components in its domain for one or more management disciplines. It, together with its operators (human or programmed), represents the final level at which network management decisions are made.

Entry Point: A network management entry point is a product or set of products that provides network management support for itself and attached products. An entry point is an SNA physical unit, and performs the network management functions of the physical unit. It transports both network management and operational data on a common SNA link. The entry point and the devices it supports must be in the same domain and network as its focal point. It uses SNA formats and protocols when communicating with its focal point.

Service Point: A network management service point is a product or set of products that provides network management support for products for which network management entry point support does not exist. It transports only network management data for these products. The service point must be in the same domain and network as its focal point. The products it is supporting need not be in the same domain or network as the service point. A service point provides a connection through which network management data can be converted to SNA formats and transmitted to the focal point for processing. It uses SNA formats and protocols when communicating with its pocal point.

A given hardware or software product may perform the focal point, entry point, or service point role, or any combination of these roles.

The relationship between entry point, service point, and focal point is often symbolized by the diagram in Figure 1 on page 3 to illustrate the relationships between the three product roles. In the Network Management Services area, facilities such as Network Management Vector Transport (NMVT) and Application Program Interface/Communications Services (API/CS) are available.

Open Communications Architectures provide documentation for SNA, applications program interfaces, and support to enable users to integrate non-SNA and/or non-IBM network components into the SNA network management environment.

Network Management Structure

The products that comprise the customer's information system are divided into three product roles: focal point, entry point, and service point. These product roles define the framework for the Network Management strategy. This structure can be applied to all components of an information network, as shown in Figure 2 on page 5, including SNA components and non-SNA components handling voice, image, data or other information.

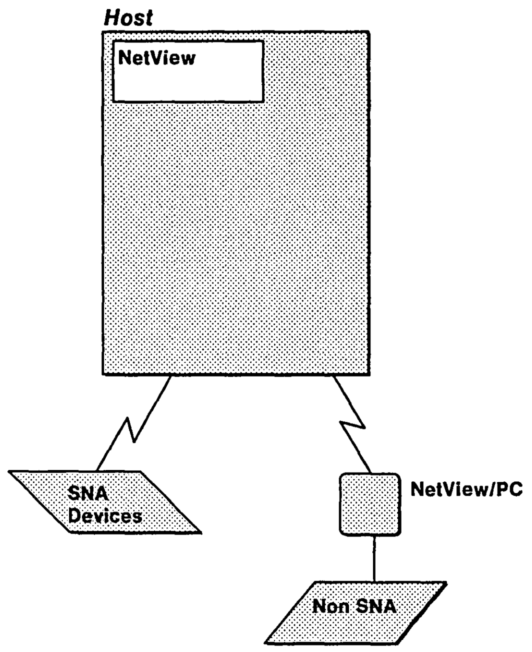


Figure 2. Network Management Structure

NetView is the primary focal point product.

Some examples of entry point products are IBM 3174, IBM 3274, IBM 3708, IBM Series/1, IBM 3720/3725, IBM System/36, and IBM System/38.

NetView/PC is an implementation of the service point.

NetView/PC in Communications Network Management

NetView/PC, as shown in Figure 3 on page 6, provides common systems services, monitoring and problem determination services, and communications channels used to transfer network management data to focal point applications. It provides for network management applications to send alerts to NetView, receive commands from NetView and respond to those commands, and to transfer data between NetView and NetView/PC.

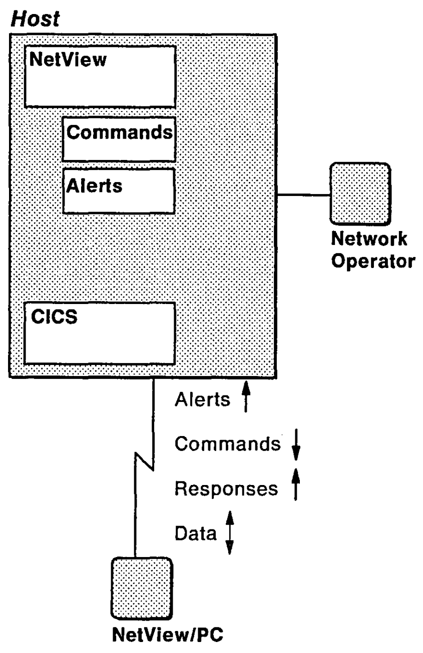


Figure 3. Network Management Products

The NetView/PC API/CS enables customers and vendors of telecommunications products to write applications which extend network management to non-IBM communications devices, as shown in Figure 4.

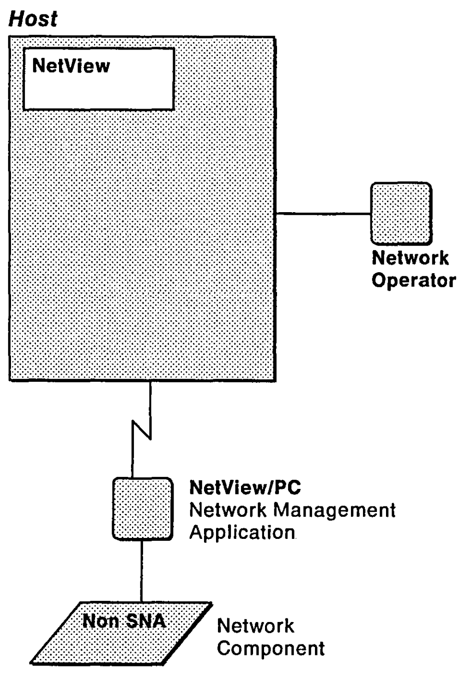


Figure 4. Network Management and Vendor Products

NetView/PC extends the Network Control Center operating area. Network management applications may use NetView/PC as a service point to extend CNM to non-SNA products.

Where Your Network Management Application Fits

Your network management application (vendor application), as shown in Figure 4 on page 6, executes as a DOS application in NetView/PC.

It will use the Application Program Interface/Communications Services (API/CS) to centralize the management of the network components managed by your application.

Chapter 2. Application Program Interface/Communications Services (API/CS)

The API/CS is provided as a NetView/PC interface to allow your network management application to centralize the management of your (vendor) product, as shown in Figure 4 on page 6.

It provides a means for your DOS Application running in the NetView/PC DOS partition in memory, to use the communication services of NetView/PC and the IBM network management facilities in NetView/PC and NetView to manage non-SNA components.

The API/CS is a 'call' interface to DOS Assembler Subroutines. The subroutine names and the function provided by each are:

DCJVA00:	Alerts
DCJVO00:	Operator Communications
DCJVC00:	Service Point Command Facility
DCJVB00:	SPCF Build and Parse
DCJVD00:	Host Data Facility

The API/CS subroutines are linked with user-written DOS applications.

The API/CS supports IBM PC DOS applications written with IBM PC Macro Assembler 2.0 language. Although any program that can be linked with the API/CS subroutines may function correctly, there is no support implied for any other language. Users who choose to use the interface for languages other than the IBM PC Macro Assembler 2.0 do so at their own risk. Problems must be recreated using the IBM PC Macro Assembler 2.0 to receive service/support from IBM.

Your network management application uses the API/CS supported NetView/PC functions while running in the NetView/PC DOS partition by calling the API/CS subroutines. You pass parameters to the subroutines in an application request block (ARB). The API/CS subroutines provide the programming interface for the NetView/PC environment.

The API/CS Subroutines provide the following four major functions. Each major function must be opened by the application before the function can be used.

Alerts: Allow an application to send alerts to NetView and/or to NetView/PC.

Operator Communications: Allow an application to turn on an icon in the icon window on line 25. The icon indicates to the operator that the DOS Command Session should be selected from the session selection panel.

Service Point Command Facility: Allow an application to receive messages from a NetView command processor and send a reply to the NetView command processor.

Host Data Facility: Allow an application to transfer (send or receive) file data to or from the Host CICS DDM application.

Using the API/CS

An Application Request Block (ARB) is required for all calls to API/CS functions. Storage for the ARB must be provided by the application program. An ARB should be dedicated to an API/CS function from the 'Open' to the 'Close' of that function.

External Declarations (such as EXTERN DCJVA00 FAR) for the API/CS library calls must *not* be in the application's code segment. All calls to API/CS subroutines are FAR calls.

To use an API/CS function, code the application to:

1. Provide storage for an ARB for each API/CS function that the API/CS will use. The storage for the ARB should be dedicated to the ARB from the 'Open' of the API/CS function to the 'Close' of the API/CS function.

   Each Application Request Block (ARB) is identified by an ARBID, ARBn, where n is a numeric character that identifies the function for which the ARB will be used. It is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump.
2. Check the address of the ARB in the AX and DX register pair when the API/CS returns control.

   The API/CS checks the ARBID and if the ARBID (ARBn) in the ARB is incorrect for the subroutine called, the ARB address is assumed to be invalid. The API/CS makes the AX and DX pair zero and returns immediately to the calling application. The application must check the AX and DX pair for non-zero before they are used.
3. Open each API/CS function that the application will use.
4. Call each API/CS function as required.
5. Close each of the API/CS functions the application has opened.

The functions should be opened as part of the application's initialization process and all opened functions should be closed by the application's termination process. The application may call an opened interface as many times as is required by the application until the application closes the (API/CS) function.

The ARB contains a 1-word (2-byte Intel Word (W)) request code field (in hexadecimal[2]) that the application sets to indicate the function desired. The request codes and descriptions are:

Request Code	Description
Alerts	
0101H	Open the Alert API/CS
0102H	Send an Alert
0104H	Close the Alert API/CS

[2] Hexadecimal (hex) representation is described in *Macro Assembler 2.0*.

Operator Communications

0201H	Open the Operator Communications API/CS
0207H	Write the icon 'DP' to the NetView/PC icon window
0208H	Clear the icon from the NetView/PC icon window
0204H	Close the Operator Communications API/CS

Service Point Command Facility

0301H	Open the SPCF API/CS
0302H	Send a RUNCMD response
0303H	Receive a RUNCMD message
0304H	Close the SPCF API/CS
0309H	Receive a command
030AH	Send a message
030BH	Send a command response
030CH	Send error sense

Host Data Facility

0401H	Open the Host Data Facility API/CS
0402H	Send file data
0403H	Receive file data
0405H	Check the status of the request
0406H	Stop file data transfer
0404H	Close the Host Data Facility API/CS

The suggested flow of an application using the API/CS follows:

```
Initialization
   .
   (User code)
   .
   Provide an ARB (storage) for each API/CS function that may be called.
   Store "ARBn" in the ARBID field.
   Set the request code to open each API/CS function
   Open each API/CS function that may be called.
   Check the AX and DX registers and the return code and take appropriate
   action.
   .
   (User code)
   .
End Initialization


application Mainline
   .
   (User code)
   .
   Store required data in the ARB
   Set the request code
   Call API/CS subroutine
   Check the AX and DX registers and the return code and take appropriate
   action
   .
   (User code)
   .
   Store required data in the ARB
   Set the request code
   Call API/CS subroutine
   Check the AX and DX registers and the return code and take appropriate
   action.
   .
   (User code)
   .
End application mainline


Termination
   .
   (User code)
   .
   Set the request code to close each API/CS function
   Close each API/CS function that is open.
   Check the AX and DX registers and the return code and take appropriate
   action.
   .
   (User code)
   .
End Termination
```

API/CS Scenario

The following scenario shows how a user-supplied DOS application, executing in the DOS partition of NetView/PC, could use IBM Open Network Management capabilities to manage a device. The scenario shows the steps relating to the DOS application from the detection of an alert condition to the transfer of file data about the device.

Figure 5 on page 14 uses numbers and arrows to show each step. It is followed by a description of what happens at each step in the diagram and refers you to the document that contains information about that particular step.

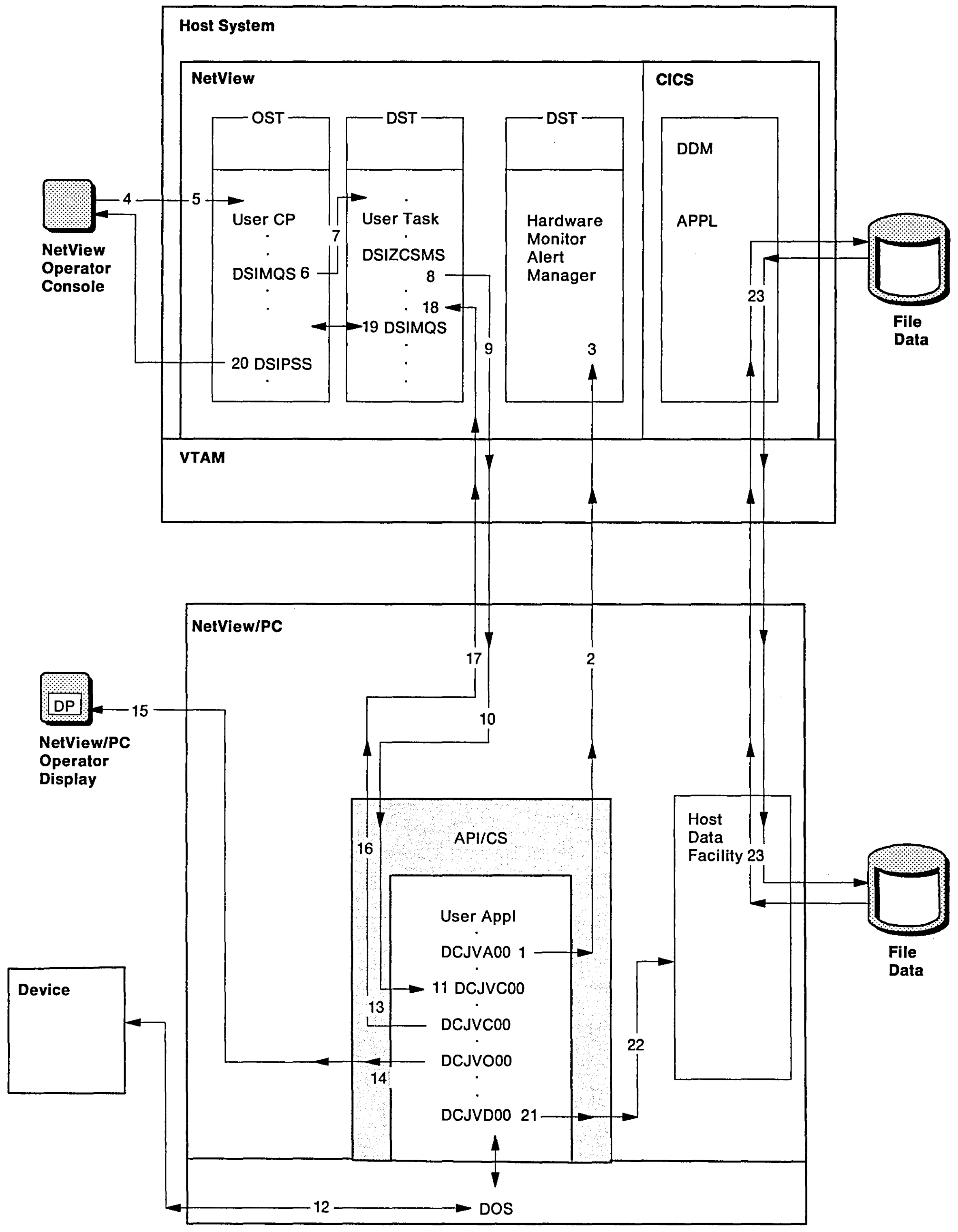


Figure 5. API/CS Network Management Scenario. Vendor network management application gain access to IBM network management services by using NetView/ API/CS. The numbers in this figure correspond to the numbered items in the scenario.

The number for the steps in Figure 5 on page 14 relate to the numbers in the following list.

An error is detected.

1. A user-supplied network management application, executing in the DOS partition of NetView/PC, recognizes an alert condition and calls DCJVA00 to send an alert to the hardware monitor and to NetView/PC.

   See Chapter 3, "Alert Subroutine Calls" on page 25 for information about writing DOS applications using the NetView/PC API/CS and constructing NetView/PC alert major vectors and subvectors.

   Also see *Systems Network Architecture Formats* about constructing an alert Network Management Vector Transport (NMVT).

2. NetView/PC logs the local alert and sends the alert NMVT to the host.

   See *IBM NetView/PC Planning and Operation Guide* for information about defining the host system to NetView/PC.

3. NetView passes the alert NMVT to the Hardware Monitor executing in NetView. See *NetView Customization* for information about alert customization.

Reacting to the Alert.

4. The NetView operator enters a network management command with the target (NetView/PC application) name.

   See *NetView Operation*.

5. The command is recognized by NetView and the Command Processor (CP) is given control and passed the parsed input.

   See *NetView Administration Reference* for information about defining user commands. See *NetView Customization* for information about adding user-supplied command processors (CP) and customizing panels.

6. The CP checks that valid data is passed and then calls the user supplied subtask with the DSIMQS macro.

   See *NetView Customization* for information about writing command processors and user subtasks and using NetView Macros in command processors and user subtasks.

7. NetView passes control to the DST with the passed data.

8. The subtask checks the input and then builds an NMVT. The subtask then sends the NMVT to NetView/PC.

9. NetView sends the NMVT to NetView/PC.

   See *NetView Customization* for information about using NetView macros.

   See *Resource Definition Guide* and *Resource Definition Reference* for information about defining NetView/PC to the host.

10. NetView/PC passes the message to the API/CS.

11. The user application calls DCJVC00 (API/CS call to receive a command) and the API/CS passes the message to the user application.

    See Chapter 5, "Service Point Command Facility (SPCF) Subroutine Calls" on page 35 and Chapter 6, "SPCF Build and Parse" on page 47 for information about NetView/PC commands and replies.

12. The user-supplied application uses DOS to perform required communications with the device.

    For information about using DOS BIOS, see *Disk Operating System.*

    See documentation supplied with the application and/or the device for information about how to control the device.

13. The user application processes the command and prepares a reply message and calls DCJVC00 (API/CS call to send a reply).

14. The user application calls DCJVO00 if necessary, to notify the operator that the DOS partition requires operator communications.

15. The API/CS turns on the DP icon on the NetView/PC operator display.

16. The API/CS sends the reply message to NetView/PC.

17. NetView/PC sends the reply NMVT to NetView.

18. NetView passes the reply NMVT to the user-supplied subtask. The subtask processes the reply and prepares a message to send to the NetView operator.

19. The command processor sends the reply to a presentation services CP.

    Note: The reply may be sent to a CLIST or may be displayed directly to the NetView operator.

20. The command processor (CP) displays the data to to NetView operator.

Transferring file data.

21. The user application calls DCJVD00 to transfer file data.

22. The API/CS passes the request to the Host Data Facility in NetView/PC.

    See *IBM NetView/PC Planning and Operation Guide* for information about the Host Data Facility. Network Manage Vector Transport (NMVT) and to build a response NMVT. See Chapter 7, "Host Data Facility Subroutine Calls" on page 57 for information about how your application can control the transfer of file data to or from the host.

23. The Host Data Transfer program initiates the transfer with the host CICS DDM application. The file data is sent to or received from the host.

    See *IBM Customer Information Control System/VS (CICS/VS)/Distributed Data Management Target Users Guide* for information about sending file data to, or receiving file data from, the host.

Programming in the NetView/PC DOS Partition

This chapter describes what the single DOS application may do while executing in the NetView/PC DOS partition in memory.

NetView/PC supports one normal PC DOS application and a large number of cooperating tasks (NetView/PC managers).

The single PC application can modify the memory allocated to it as designated in its Program Segment Prefix. It **must not** modify the memory associated with the active screen buffer. No other memory may be modified. This restriction precludes the execution of some PC applications in a NetView/PC environment. Some examples of programs that execute successfully in the DOS partition and programs that violate these restrictions are listed in "Verified PC DOS Applications" on page 19 and "Applications Not Successfully Executed" on page 19, respectively.

You should tailor interrupt handlers for the NetView/PC environment. The DOS application may take over the first asynchronous (CH) and timer (8H) hardware interrupts reliably. However, timer interrupts must be passed on to NetView/PC at the normal rate. PC DOS and BIOS service interrupt vectors must not be taken over by predecessors to NetView/PC unless the interrupt handlers are reentrant.

The single PC DOS application may use software interrupts to access the disk, keyboard, and display.

The multi-tasking environment of NetView/PC requires that the handling of DOS critical errors be modified. In DOS, critical errors are handled by DOS and the return codes for these particular errors are not normally returned to calling programs. See "DOS Error Codes" on page 71 for a list of the error codes returned, and their meaning.

When designing applications intended to execute in the DOS partition of NetView/PC, give special consideration to the following:

- Control characters are not processed as they are by DOS when in a DOS Compatibility session with the session manager.
- If a DOS session is aborted because of a critical error (such as divide overflow) all other DOS sessions will be locked out.

Requirements and Restrictions

To use the API/CS the following requirements and restrictions must be satisfied.

1. Timer interrupts must be passed on to NetView/PC at the normal rate. See "You should tailor interrupt handlers . . ." on page 17
2. PC DOS and BIOS service interrupt vectors must not be taken over. See "PC DOS and BIOS service . . ." on page 17
3. Application programs must only make calls to the API/CS subroutines while executing in the DOS partition of NetView/PC. Calls to the API/CS subroutines in a native DOS environment will 'lock up' the PC. To recover, the PC must be powered off and then back on.
4. The DOS partition restrictions in the *IBM NetView/PC Planning and Operation Guide* must be followed.
5. Software interrupts X'75' and X'78' through X'7F' are currently used by NetView/PC and must not be modified.
6. The DOS application must provide 100 bytes of "STACK" space for the API/CS.
7. The DOS application must use DOS or "BIOS" calls for video and keyboard I/O.
8. The DOS application must fit in the DOS partition of the NetView/PC grouping in which it is intended to run.
9. The appropriate subroutines that provide the desired functions must be linked with the DOS application. See *Macro Assembler 2.0*.
10. The application must allocate storage for and construct an Application Request Blocks (ARB) for each interface used.

11. The application must pass the address of the start of the Application Request Block in the AX and DX register pair when the API/CS subroutine is called. The Segment Address must be in the AX register. The offset address must be in the DX register.
12. All calls to the API/CS subroutines must be FAR calls.
13. NMVT fields that specify character data must be filled in as ASCII character data by the application program.
14. The AX and DX register pair must be checked on return from a call to the API/CS. All registers are saved by the API/CS and restored on return to the application from the API/CS **except** AX and DX. The AX and DX are **zero** if the ARBID is incorrect for the call. On return from a call to the API/CS, the application must check that they are non-zero before they are used.
15. Request codes must be coded in hexadecimal.
16. Message file names must be in the form "cccc.MSG" as required by EZ–VU, where cccc is a four-character name and MSG is the extension.
17. NetView/PC panels must not be altered with EZ–VU II.
18. The EZ–VU II configuration utility panel 5 variable 'ZDBW' must be 'N'.

Installation

API/CS modules are shipped with and are part of NetView/PC. Link user-supplied applications with the NetView/PC library containing the API/CS modules.

The DOS Linker is used to link the user object modules with APICS.LIB.

Using EZ – VU

If you use *EZ – VU II Development Facility for the IBM PC.* (EZ–VU), your application must not change environment variables ISPPRO, ISPPGM, ISPMSG, and ISPPAN.

EZ – VU Calls

Only the BP and DS registers are saved by EZ–VU. To call EZ–VU:

1. Save the programs registers on the stack.
2. Save the SP register in BP.
3. Push EZ–VU parameters onto the stack.
4. Call EZ–VU.

On return from EZ–VU:

1. Restore SP from BP.
2. Restore the other saved registers from the stack.

"EZ – VU Calls" shows how the PC DOS sample program saves and restores registers when calls to EZ–VU are made.

Directories

All EZ–VU files except panels must be in the NETVIEW subdirectory.

All program panel files must be in the NVPCPANL subdirectory. Also copy EZ–VU panel ISPFMNT1.PAN to the NetView/PC subdirectory NVPCPANL.

Verified PC DOS Applications

To test that existing DOS applications can execute successfully in the NetView/PC DOS partition in memory, a limited number of DOS applications have been executed successfully in the DOS partition.

The following PC DOS applications have executed successfully in the DOS partition of NetView/PC:

- DOS commands
  - CHKDSK
  - FORMAT
  - PC DOS Piping and the MORE command.
  - SORT requires at least 66K for execution.
  - TREE
- Personal Productivity applications
  - EZ–VU II, a dialogue manager for the PC, similar to ISPF

    Note: You must not modify NetView/PC panels using EZ–VU II.
  - IBM File List (if loaded after NetView/PC. File List has no effect if loaded before NetView/PC).
  - Visicalc
- Communications packages.
  - IBM 3101 Emulator
- Compilers and system tools.
  - IBM Macro Assembler.
  - IBM Pascal Compiler.
  - CI86 'C' Compiler from Computer Innovations.

Applications Not Successfully Executed

Only a limited number of programs have been tested. Of those programs, the following programs did not execute successfully, in the DOS partition of NetView/PC, as the single DOS application:

- VMPC (does not respect memory regions)
- Time Manager (does not respect memory regions)
- Snipes (does not respect memory regions)
- IBM Professional Editor (intercepts the keyboard interrupt).

IBM Personal Editor writes directly to video memory. It may be used with little impact to NetView/PC because it performs video updates only as a result of keyboard input. Keyboard input only occurs (for this program) while it is the selected (foreground) task.

Relating to Documentation

Figure 6 on page 21 shows the relationships of an application program executing in the DOS partition of NetView/PC and using the API/CS, to the elements of Open Communication Architectures (OCA) in NetView/PC and in the host. The numbers in the figure correspond to documentation list number for the numbered function or facility.

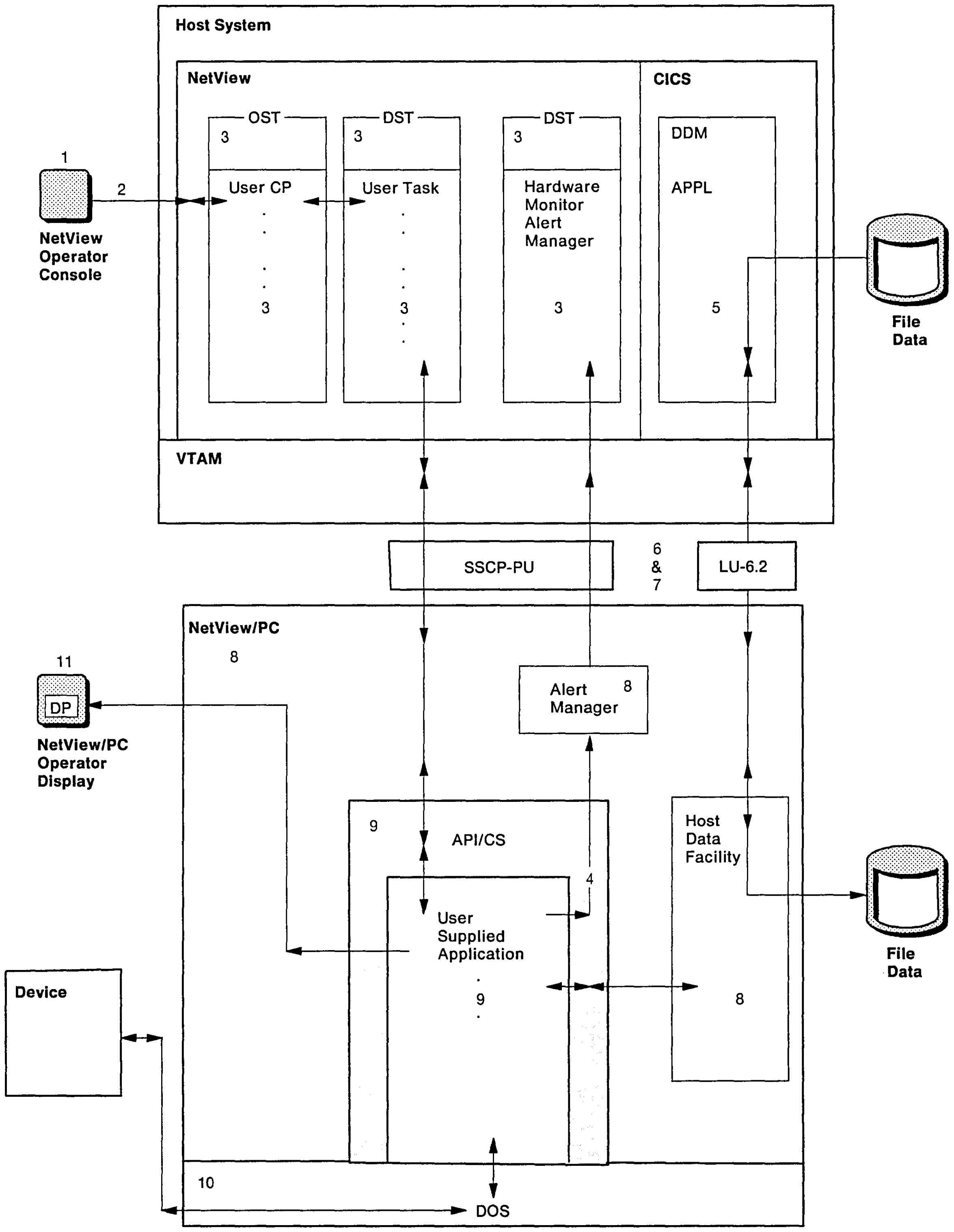


Figure 6. Documentation for the Environment. The numbered list items correspond to the numbers in this figure.

1. For information about operating NetView, see *NetView Operation*.
2. For information about defining user commands, see *NetView Administration Reference*.

3. For information about writing and adding user-supplied command processors (CP) and subtasks, and customizing panels, see *NetView Customization*.
4. For information about Network Management Vector Transport (NMVT) structures, see *System Network Architecture Formats*.
5. For information about the CICS DDM application, see *IBM Customer Information Control System/VS (CICS/VS)/Distributed Data Management Target Users Guide*.
6. For information about defining NetView/PC to the host system, see *NetView Installation and Administration Guide*, *Resource Definition Guide*, and *Resource Definition Reference*.
7. For information about defining the host to NetView/PC, see *IBM NetView/PC Planning and Operation Guide*.
8. For information about NetView/PC, see *IBM NetView/PC Planning and Operation Guide*. This guide provides information about the Host Data Facility and the Alert Manager.
9. For information about writing DOS applications to use the NetView/PC API/CS, this book describes how to:
    a. Send alerts to NetView.
    b. Notify the NetView/PC operator that the application executing in the DOS partition requires operator communications.
    c. Receive command messages from a host operator and send reply messages back to the operator.
    d. Parse commands received from NetView and build replies to the commands
    e. Transfer data between the NetView/PC and the host CICS DDM application.
10. For information about DOS, see *Disk Operating System*.
11. For information about using EZ–VU II for dialog management, see *EZ–VU II Development Facility for the IBM PC*.

Part 2. Using the API/CS Subroutines

Chapter 3. Alert Subroutine Calls 25
Alert ARB 26
Primary Alert API/CS Return Codes 27
Secondary Alert API/CS Return Codes 27
Open the Alert API/CS 30
Send an Alert 30
Close the Alert API/CS 30

Chapter 4. Operator Communications Subroutine Calls 31
Operator Communications ARB 31
Operator Communications API/CS Return Codes 32
Open the Operator Communications API/CS 32
Write the Icon 'DP' to the NetView/PC Icon Window 32
Clear the Icon from the NetView/PC Icon Window 33
Close the Operator Communications API/CS 33

Chapter 5. Service Point Command Facility (SPCF) Subroutine Calls 35
SPCF ARB 37
SPCF API/CS Return Codes 39
Open the SPCF API/CS 40
Receive a RUNCMD message 40
Send a RUNCMD response 41
RUNCMD Response Message Buffer 42
Send a Message 42
Receive a Command 43
Send a Command Response 43
Send Error Sense 44
Defined SENSETYPE values 44
Defined LCCSTAT Values 45
Defined Error Detail Values 45
Close the SPCF API/CS 45

Chapter 6. SPCF Build and Parse 47
Parse 47
Parse SPCF Command ARB 47
Parse SPCF Command API/CS Return Codes 49
Parse Request 49
Returned ARB Data Fields 50
Parse Sense Data Definitions 50
Names List Format 50
Build 51
Build SPCF Reply ARB 51
Build SPCF Reply API/CS Return Codes 52
Build Request 53
Link Status Value Definitions 54
Path Information List Control Blocks 54
LINKPD LCC Description Control Block 54
LINKDATA And LINKTEST LCC Description Control Block 55
LINKDATA And LINKTEST LCC Data Control Block 55
LCC Data 56
Probable Cause 56

Chapter 7. Host Data Facility Subroutine Calls 57
Host Data Facility ARB 57
Host Data Facility API/CS Return Codes 58
Open the Host Data Facility API/CS 60
Send File Data 60
Receive File Data 61
Check the Status of a Host Data Facility Request 61
Stop File Data Transfer 61
Close the Host Data Facility API/CS 62

Chapter 3. Alert Subroutine Calls

NetView/PC™ V1.1 supports non-generic, generic, and hybrid alerts. Generic alerts can only be sent to NetView™ Release 2 and do not require stored screen support. Non-generic and hybrid alerts can be sent to the NetView/PC Alert Manager and/or NetView Release 1 or NetView Release 2. See Appendix B, "Alert Major Vector Formats" on page 77 for information about how to build non-generic, generic, and hybrid alert NMVTS. The alert subroutine provides for the transportation of alert data to the NetView/PC Alert Manager and/or NetView. The application program using the API/CS is responsible for ensuring that the alert major vectors and subvectors are correct.

When the NetView/PC Alert Manager has a session with NetView, and the application requests it, the alert will be sent to NetView by the NetView/PC Alert Router. The alert NMVT character data fields will be translated from ASCII to EBCDIC by NetView/PC before it is sent to NetView. The Alert API/CS request codes and descriptions are:

0101H	Open the Alert API/CS
0102H	Send an Alert
0104H	Close the Alert API/CS

To use the API/CS to send alert data to the NetView/PC Alert Manager, the application must provide memory for and create an ARB. The following API/CS calls must then be coded:

1. Call DCJVA00 with request code 0101H to open the Alert API/CS.
2. Call DCJVA00 with request code 0102H to send the application alert data to NetView and/or NetView/PC.
3. Call DCJVA00 with request code 0104H to close the Alert API/CS.

Extensive checking of the alert NMVT is done by NetView/PC to ensure that the alert NMVT is correct before it is sent. The checks provide return code (RC), error class, and error type information that can be used for debugging during application development. The error indications are provided in the ARB. The ARB contains primary and secondary RC, error class, and error type fields. When the primary RC, class, or type is non-zero, the secondary error RC, class, and type fields should be checked to determine the cause of the error indication. The Alert Router secondary codes identify problems with the alert NMVT syntax before the alert NMVT is sent to the host.

Be sure that support for the alerts you plan to send is provided in NetView, either by IBM-provided support or by user-defined alerts. NetView/PC will not reject alerts that are not supported by the receiving NetView.

NetView supports non-generic alerts in one of the four following ways:

1. Default support — hexadecimal display of alert subvectors
2. IBM stored screen support — formatted displays shipped with NetView
3. Modified screen support — user-modified formatted displays
4. User-defined — formatted displays defined by the user.

User exits may also be used to display alert data.

Alert ARB

The format of the Alert ARB, and a description of the ARB fields follows:

Disp	Lgth	Name	Description
0	04	ARBID	A 4-character constant that is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump. The 4-character constant 'ARB1' must be stored in the ARBID field.
4	02	REQUEST CODE	A word (2-byte Intel Word (W)) request identifier. Each request has a unique code that must be stored in the ARB by the Application. The first byte identifies the function and the second byte identifies the request.
6	01	ARB LENGTH	The length (44) of the ARB for this API/CS function. The length must be stored into the ARB by the application.
7	02	Reserved	Reserved and must be initialized to binary zeros.
9	02	Return Code	An indicator of the degree of success in performing the request.
11	02	Class	The error class.
13	02	Type	The error-type.
15	04	MVADDR	A 4-byte (word offset and word segment) address pointing to a buffer that contains the Alert major vector that is to be sent to NetView. See Appendix B, "Alert Major Vector Formats" on page 77.
19	01	MVTARG	(B|H|L) Character (1) keyword that indicates whether the Alert is to be sent to the local (L) network manager (NetView/PC), to the host (H) network manager (NetView), or to both (B). Defaults to B if not specified or if an invalid value is specified.
			Secondary return code, class, and type are in the following fields. They are an indicator of the degree of success of the functions used by the API/CS in performing the users request. They are provided for problem determination and problem isolation of problems experienced by the users of the API/CS. When the primary RC, class, and type are non-zero, check the secondary RCs, classes, and types, and take appropriate action.
20	02	Alert RC	Return code from the NetView/PC Alert Manager.
22	02	Alert Error Class	Error class from the NetView/PC Alert Manager.
24	02	Alert Error Type	Error type from the NetView/PC Alert Manager.
26	02	Alert Router RC	Return code from the Alert Router.
28	02	Alert Router Error Class	Error class from the Alert Router.
30	02	Alert Router Error Type	Error type from the Alert Router.
32	02	Host RC	Return code about host communications.
34	02	Host Error Class	Error class about host communications.
36	02	Host Error Type	Error type about host communications.
38	02	Reserved	Reserved

Figure 7 (Part 1 of 2). Alert ARB

Disp	Lgth	Name	Description
40	02	Reserved	Reserved
42	02	Reserved	Reserved

Figure 7 (Part 2 of 2). Alert ARB

Primary Alert API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0008	0001	0047	Invalid request
0008	0002	0009	Storage not available
0008	0008	0008	Unexpected error. See other return codes for furtherexplanation
0008	0008	0096	NetView/PC Alert Manager not available
0008	0012	0096	NetView/PC Alert Manager and host session are not available
0008	0017	0070	The function has already been opened
0008	0065	0070	The function has not been opened
0008	0096	0098	Alert Router is currently not available
0008	0098	0096	Host session not available
0008	0117	0115	Request processed without error for NetView/PC Alert Manager, but did not process for host
0008	0117	0116	Request processed without error for host, but did not process for, or received a warning from, the NetView/PC Alert Manager

Figure 8. Primary Alert API/CS Return Codes

Secondary Alert API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0008	0001	0019	Invalid NMVT length
0008	0001	0023	Invalid NMVT key field
0008	0001	0024	File write access locked
0008	0001	0026	Invalid record (journal)
0008	0001	0040	Date/Time subvector data invalid
0008	0001	0041	Basic subvector data invalid
0008	0001	0042	PSID subvector data invalid
0008	0001	0043	Hierarchy Names subvector data invalid
0008	0001	0044	NetView/PC Alert subvector data invalid
0008	0001	0045	Text subvector data invalid
0008	0001	0136	Invalid character for ASCII to EBCDIC translation
0008	0001	0144	Detail qualifier subvector data invalid

Figure 9 (Part 1 of 3). Secondary Alert API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0001	0147	LAN subvector data invalid
0008	0002	0040	Date/Time subvector missing
0008	0002	0041	Basic subvector missing
0008	0002	0042	PSID subvector missing
0008	0002	0043	Hierarchy Names subvector missing
0008	0002	0044	NetView/PC Alert subvector missing
0008	0002	0160	Hierarchy resource list subvector missing
0008	0002	0162	Link station data subvector missing
0008	0002	0163	Generic alert data subvector missing
0008	0002	0164	Probable cause subvector missing
0008	0002	0165	User cause subvector missing
0008	0002	0166	Install cause subvector missing
0008	0002	0167	Failure cause subvector missing
0008	0002	0168	Undetermined cause subvector missing
0008	0002	0169	Detailed data subvector missing
0008	0002	0170	Self-defining text message subvector missing
0008	0003	0040	Duplicate Date/Time subvector
0008	0003	0041	Duplicate Basic subvector
0008	0003	0042	Duplicate PSID subvector
0008	0003	0043	Duplicate Hierarchy Names subvector
0008	0003	0044	Duplicate NetView/PC Alert subvector
0008	0003	0045	Duplicate Text subvector
0008	0003	0144	Duplicate Detail Qualifier subvector
0008	0003	0147	Duplicate LAN subvector
0008	0003	0160	Duplicate Hierarchy resource list subvector
0008	0003	0162	Duplicate Link station data subvector
0008	0003	0163	Duplicate Generic alert data subvector
0008	0003	0164	Duplicate Probable cause subvector
0008	0003	0165	Duplicate User cause subvector
0008	0003	0166	Duplicate Install cause subvector
0008	0003	0167	Duplicate Failure cause subvector
0008	0003	0168	Duplicate Undetermined cause subvector
0008	0003	0169	Duplicate Detailed data subvector
0008	0003	0170	Duplicate Self-defining text message subvector
0008	0008	0008	Unexpected error. See other return codes for furtherexplanation
0008	0008	0023	Major vector key field format error
0008	0008	0040	Date/Time subvector format error
0008	0008	0041	Basic subvector format error
0008	0008	0042	PSID subvector format error
0008	0008	0043	Hierarchy Names subvector format error
0008	0008	0044	NetView/PC Alert subvector format error
0008	0008	0045	Text subvector format error

Figure 9 (Part 2 of 3). Secondary Alert API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0008	0096	NetView/PC Alert Manager not available
0008	0008	0144	Detail Qualifier subvector format error
0008	0008	0147	LAN subvector format error
0008	0008	0160	Hierarchy resource list subvector format error
0008	0008	0162	Link station data subvector format error
0008	0008	0163	Generic alert data subvector format error
0008	0008	0164	Probable cause subvector format error
0008	0008	0165	User cause subvector format error
0008	0008	0166	Install cause subvector format error
0008	0008	0167	Failure cause subvector format error
0008	0008	0168	Undetermined cause subvector format error
0008	0008	0169	Detailed data subvector format error
0008	0008	0170	Self-defining text message subvector format error
0008	0012	0068	File I/O error
0008	0065	0078	CP-PU not active; retry
0008	0098	0009	Storage not available
0008	0098	0068	Security file not available
0008	0098	0096	Host session not available
0008	0159	0002	Dependent key missing
0008	0159	0023	Key dependency error
0008	0159	0040	Date/Time subvector key dependency error
0008	0159	0041	Basic subvector key dependency error
0008	0159	0042	PSID subvector key dependency error
0008	0159	0043	Hierarchy Names subvector key dependency error
0008	0159	0044	NetView/PC Alert subvector key dependency error
0008	0159	0045	Text subvector key dependency error
0008	0159	0144	Detail Qualifier subvector key dependency error
0008	0159	0147	LAN subvector key dependency error
0008	0159	0160	Hierarchy resource list subvector key dependency error
0008	0159	0162	Link station data subvector key dependency error
0008	0159	0163	Generic alert data subvector key dependency error
0008	0159	0164	Probable cause subvector key dependency error
0008	0159	0165	User cause subvector key dependency error
0008	0159	0166	Install cause subvector key dependency error
0008	0159	0167	Failure cause subvector key dependency error
0008	0159	0168	Undetermined cause subvector key dependency error
0008	0159	0169	Detailed data subvector key dependency error
0008	0159	0170	Self-defining text message subvector key dependency error

Figure 9 (Part 3 of 3). Secondary Alert API/CS Return Codes

Open the Alert API/CS

Purpose: To allow an application to use the API/CS to send alerts to NetView.

Setting Up:

1. Provide memory for an ARB.
2. Store "ARB1" in the ARBID field of the ARB.
3. Store request code 0101H in the request code field of the ARB.
4. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVA00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Send an Alert

Purpose: To send an alert to NetView/PC and/or to NetView.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Provide memory for a buffer.
3. Format the alert data in the buffer as an alert major vector. See Appendix B, "Alert Major Vector Formats" on page 77 for building NetView/PC alerts.
4. Store the address of the alert major vector in the MVADDR field of the ARB.
5. Store B (both), H (host), or L (local) in the MVTARG field of the ARB.
6. Store request code 0102H in the request code field of the ARB.
7. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVA00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Close the Alert API/CS

Purpose: To terminate the use of the send alert function of the API/CS.

Setting Up:

1. Store request code 0104H in the request code field of the ARB.
2. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVA00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Chapter 4. Operator Communications Subroutine Calls

The Operator Communications (OC) API/CS allows an application program to turn on icon "DP" in the NetView/PC icon window to indicate that the DOS partition should be selected. The application must turn off the icon when the purpose for turning the icon 'on' is no longer valid. The icon will stay 'on' until it is turned 'off' by the application or until the Operator Communications API/CS is closed.

The request codes used for the Operator Communications API/CS and descriptions are:

0201H	Open the Operator Communications API/CS
0207H	Write the icon 'DP' to the NetView/PC icon window
0208H	Clear the icon from the NetView/PC icon window
0204H	Close the Operator Communications API/CS

To use the API/CS to turn on the DOS Partition icon in the icon window of the operator display, the following API/CS calls must be coded:

1. Call DCJVO00 with request code 0201H to open the Operator Communications API/CS.
2. Call DCJVO00 with request code 0207H to Write the icon 'DP' to the NetView/PC icon window
3. Call DCJVO00 with request code 0208H to clear the icon.
4. Call DCJVO00 with request code 0204H to close the Operator Communications API/CS when there is no more need for the "DP" icon on the NetView/PC operator display to be on.

Operator Communications ARB

The format of the Operator Communications ARB, and a description of the ARB fields follows:

Disp	Lgth	Name	Description
0	04	ARBID	A 4-character constant that is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump. The 4-character constant 'ARB2' must be stored in the ARBID field.
4	02	REQUEST CODE	A word (2-byte Intel Word (W)) request identifier. Each request has a unique code that must be stored in the ARB by the Application. The first byte identifies the function and the second byte identifies the request.
6	01	ARB LENGTH	The length (15) of the ARB for this API/CS function. The length must be stored into the ARB by the application.
7	02	Reserved	Reserved and must be initialized to binary zeros.
9	02	Return Code	An indicator of the degree of success in performing the request.
11	02	Class	The error class.
13	02	Type	The error type.

Figure 10. Operator Communications ARB

Operator Communications API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0008	0001	0047	Invalid request
0008	0002	0009	Storage not available
0008	0017	0070	The function has already been opened
0008	0065	0070	The function has not been opened

Figure 11. Operator Communications API/CS Return Codes

Open the Operator Communications API/CS

Purpose: To allow an application to use the API/CS to control the DOS icon "DP" on line 25 of the NetView/PC display.

Setting Up:

1. Provide memory for an ARB.
2. Store "ARB2" in the ARBID field of the ARB.
3. Store request code 0201H in the request code field of the ARB.
4. Set the segment and offset register pair (AX-DX) to point to the start of the ARB.

CALL DCJVO00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Write the Icon 'DP' to the NetView/PC Icon Window

Purpose: To allow the application to turn on the DOS Partition icon "DP" in the icon window on line 25 of the NetView/PC operator display panel.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Store request code 0207H in the request code field of the ARB.
3. Set the segment and offset register pair (AX-DX) to point to the start of the ARB.

CALL DCJVO00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Clear the Icon from the NetView/PC Icon Window

Purpose: To allow the application to turn off the DOS Partition icon "DP" in the icon window on line 25 of the NetView/PC operator display panel.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Store request code 0208H in the request code field of the ARB.
3. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVO00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Close the Operator Communications API/CS

Purpose: To terminate the use of the Operator Communications function of the API/CS.

Setting Up:

1. Store request code 0204H in the request code field of the ARB.
2. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVO00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Chapter 5. Service Point Command Facility (SPCF) Subroutine Calls

NetView/PC™ API/CS provides the capability for application programs executing in the DOS partition in NetView/PC, to:

- Receive any unparsed command from NetView and respond to the command
- Send unsolicited messages to a NetView operator
- Receive a RUNCMD message from a NetView operator and respond to the message

See Chapter 6, "SPCF Build and Parse" on page 47 for a description of the parse and build facilities provided by API/CS for the following NetView™ Release 2 commands:

- LINKDATA
- LINKPD
- LINKTEST
- RUNCMD

The NetView commands are described in *NetView Operation*, SC30-3364. The API/CS supported commands are also described in "API/CS Supported NetView Commands" on page 89.

When applications use the API/CS to receive a supported NetView command, the unparsed command NMVT is passed to the application by the SPCF API/CS subroutine. The application must interpret the meaning of the received command and construct an NMVT to respond to the command.

The SPCF Subroutine will support applications written for the version 1.0 ARB. It will use additional request codes and ARB fields to support the new version 1.1 functions.

API/CS has provided Build and Parse requests to help interpret received commands and to construct response NMVTs. For information about using the Build and Parse API/CS functions, see Chapter 6, "SPCF Build and Parse."

The API/CS also provides for the transportation of messages from and replies to a user-supplied Data Services Task (DST) invoked from an Operator Services Task (OST) running under NetView in the host.

The request codes used by the SPCF API/CS and descriptions are:

0301H	Open the SPCF API/CS
0302H	Send a RUNCMD response
0303H	Receive a RUNCMD message
0304H	Close the SPCF API/CS
0309H	Receive a command. An unparsed command NMVT, if present, is returned. The application is required to parse the NMVT to determine the command.

030AH	Send a message. An unsolicited message is sent to a NetView operator from a file or from a buffer.
030BH	Send a command response. The response NMVT is sent to NetView as received from the application. The application is required to format the NMVT.
030CH	Send error sense The application has the option of sending error sense data provided by NetView/PC or sending user-defined error sense data.

When each command or message is received, a correlator is returned to the application in the Recvcorr field of the ARB. The correlator of the message must be stored in the SENDCORR field of the ARB when responses are sent. Up to eight (8) commands may be received before the application must send a response. The application must save the correlator for each command and ensure that the correct correlator is used for the response.

The application program using the API/CS is responsible for ensuring that the response correlator (SENDCORR field in the ARB) matches the command that is being responded to, and for ensuring that the response data text is correct. See *Resource Definition Guide, Resource Definition Reference, IBM NetView/PC Planning and Operation Guide*, and *NetView Administration Reference* for a description of requirements to communicate with NetView.

RUNCMD response messages may be contained in message files which conform to the file and message format of EZ−VU messages or may be passed from the DOS application to the API/CS to be sent to NetView.

RUNCMD response messages to be sent from a message file must be in the same subdirectory with the NetView/PC message file. The file name is in the form cccc.MSG. See *EZ−VU II Development Facility for the IBM PC.*

Message files may be created with most popular IBM PC editors.

A message must begin with the 4-character numeric message identifier terminated with a blank. The blank may be followed by up to 65 characters of text terminated by the string X'0D0A' (carriage return, line feed).

RUNCMD response messages from the application are passed to the API/CS in a buffer. The application specifies whether the message data is to be translated from ASCII to EBCDIC before it is sent. If translation is not requested, only one message of up to 478-bytes may be sent. If translation is requested, several messages may be put in the one 478-byte message buffer. See "Translation of NMVT Data Fields" on page 72 for a description of the translation performed.

Use the physical unit (PU) name for NetView/PC as the service point name to send messages or commands to the target NetView/PC.

The applications using the API/CS subroutines must also open the SPCF API/CS with a name known to programs and/or operators that will be communicating with the applications.

To send unsolicited messages to an operator, the application must know the operator's NetView logon name.

To use the API/CS to receive commands and messages from a NetView operator and send messages or respond to commands, code the application to perform the following steps and subroutine calls:

1. Construct an ARB with ARB LENGTH set to 90[3]
2. Call DCJVC00 with request code 0301H to open the Service Point Command Facility API/CS.
3. Call DCJVC00 with the appropriate receive request code
   - Receive a RUNCMD message
   - Receive a command
4. Store the correlator of the received command in the SENDCORR field of the ARB.
5. Call DCJVC00 with the appropriate request code to send a response
   - Send a RUNCMD response message
   - Send a command response
   - Send error sense
6. Call DCJVC00 with request code 0304H to close the SPCF API/CS

Note: If data is required to be sent to the host in a format not supported by the RUNCMD, an Operator Services Task (OST) and Data Services Task (DST) can be written and installed on NetView to provide the unique support required. See the sample programs Appendix J, "NetView Sample Programs" for guidance on how to provide the unique support. Required resources must have been defined (see "Where to Find More Information" on page vi) whether the RUNCMD is used or user-supplied command processors are used.

SPCF ARB

The format of the SPCF ARB, and a description of the ARB fields follows:

Disp	Lgth	Name	Description
0	04	ARBID	A 4-character constant that is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump. The 4-character constant 'ARB3' must be stored in the ARBID field.
4	02	REQUEST CODE	A word (2-byte Intel Word (W)) request identifier. Each request has a unique code that must be stored in the ARB by the Application. The first byte identifies the function and the second byte identifies the request.
6	01	ARB LENGTH	The length (90) of the ARB for this API/CS function. The length must be stored into the ARB by the application. The length must be 90 if request codes 0309H, 030AH, 030BH, and 030CH will be used. The length may be 67 if only request codes 0301H, 0302H, 0303H, and 0304H will be used.
7	01	PARSE ID	A 1-byte field returned by the API that contains the least significant byte of the major vector (MV) key of the command NMVT.
8	01	Reserved	Reserved and must be initialized to binary zeros.
9	02	Return Code	An indicator of the degree of success in performing the request.
11	02	Class	The error class.
13	02	Type	The error type.
15	08	TARGET NAME	A 1 to 8-character application name that the application is known as.

Figure 12 (Part 1 of 3). SPCF ARB

[3] ARB LENGTH may be set to 67 for Receive a RUNCMD call.

Disp	Lgth	Name	Description
23	01	MSGTYPE	(B|F) Character (1) keyword that indicates whether the message data to be sent is in a buffer or is in a message file. When MSGTYPE = 'B', the message data to be sent is in a buffer. When MSGTYPE = 'F', the message data to be sent is in a file.
24	04	Msgfile	When MSGTYPE = 'F', Msgfile contains the 4-character name of the message file that contains the message to be sent to NetView. The 4-character file name must be in the form required by EZ-VU.
28	04	Msgnum	A 4-character numeric message identifier of the reply message in the file named in the Msgfile field. The 4-character message identifier must be in the form required by EZ-VU. Leading character zeros are required for numbers less than 4 characters long. Message data from the file is translated from ASCII to EBCDIC before it is sent. Must be zero if no message data is to be sent from a file or if message data to be sent is contained in a buffer pointed to by Msgbuff.
32	02	MBlength	A word (16-bit integer) length of the data to be sent from the buffer pointed to by the Msgbuff field. Must be equal to or less than 473 if Convert is 'N'. Not examined if Convert is 'Y' because the length is computed from the message list lengths and the Msgcount. Not examined for Send a Command Response (030BH) and Send Error Sense (030CH) requests.
34	02	Msgcount	A word (16-bit integer) count of the messages to be sent from the message buffer pointed to by the Msgbuff field. Must be zero if message data to be sent is contained in a message file. Must be one if the Convert field is 'N'.
36	01	Convert	(N|Y) Character (1) keyword that indicates whether RUNCMD response message data is to be translated from ASCII to EBCDIC before it is sent, or not (N) translated. The NetView RUNCMD will not handle unconverted ('N') reply messages. When Convert = 'N', the message data will be sent as is. Anything other than 'Y' will cause the data NOT to be translated (default to 'N'). This field is only used for 0303H requests.
37	04	Msgbuff	A 4-byte (word offset and word segment) address pointing to a buffer that contains message data to be sent.
41	01	Cmdlgth	A 1-byte length of the received message and pointed to by the Command field of this ARB. The command length is set to 0 for a Receive a Command (0309H) request and the application must parse the NMVT to get the length.
42	04	Command	A 4-byte (word offset and word segment) address pointing to a buffer that contains the received message. The area size is 256 bytes if the command received is a RUNCMD, otherwise the size is 512 bytes.
46	10	Recvcorr	A 10-byte hex correlator. The unique correlator of the last message returned for a receive call. It must be stored in the SENDCORR field of the ARB when the reply is sent.
56	10	SENDCORR	The 10-byte correlator of the message this send reply call is replying to. The correlator is used to associate the reply message with the received message (Required for send calls). This field is ignored with 'Send a Message' (030AH) requests.
66	01	Force	(N|Y) Character (1) keyword used with CLOSE that indicates whether messages and commands destined for the application will be discarded. 'Y' causes queued messages and commands to be discarded and error sense is sent to the host by the API/CS. Anything other than 'Y' returns a return code.

Figure 12 (Part 2 of 3). SPCF ARB

Disp	Lgth	Name	Description
			The following fields are used with request codes 0309H, 030AH, 030BH, and 030CH.
67	08	Operator Name	An 8-Character name of the NetView Operator who will receive the unsolicited message.
75	02	Putreply length	A word (16-bit integer) length of the overall NMVT to be sent to the NetView Host. The size must not exceed 504 bytes.
77	04	Putreply	A 4-byte (word offset and word segment) address pointing to a buffer that contains the reply NMVT to be sent to the NetView Host. Used when the application chooses to send a response NMVT to the Host.
81	01	SENSETYPE	A 1-byte (8-bit integer) value that determines the sense code that will be sent back to the NetView Host when the Send Error Sense (X'030C') request code is used. This field is required for the Send error sense request. See Figure 15 on page 44 for values.
82	01	LCCSTAT	A 1-byte (8-bit integer) value of the secondary sense code that will be sent back to the NetView Host when the Send error sense (X'030C') request code is used. This field is optional for the Send error sense request and must be set to X'00' if not used. See Figure 16 on page 45 for values.
83	01	Error Detail	A 1-byte (8-bit integer) value of the error detail that will be sent back to the NetView Host when the Send error sense (X'030C') request code is used. This field is optional for the Send error sense request and must be set to X'00' if not used. See Figure 17 on page 45 for values.
84	04	User sense	A 4-byte binary user string sense code. The sense code must conform to SNA sense codes. This field is used if the SENSETYPE is 0.
88	01	SV Key	A 1-byte binary field to put the key of the subvector with the error in. This field is optional and must be set to 0 if not used.
89	01	SF Key	A 1-byte binary field to put the key of the subfield with the error in. This field is optional and must be set to 0 if not used.

Figure 12 (Part 3 of 3). SPCF ARB

SPCF API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0002	0000	0000	SPCF Request Queue is empty
0008	0001	0019	Invalid NMVT length
0008	0001	0047	Invalid request
0008	0001	0072	Invalid MSGTYPE
0008	0002	0068	File not found
0008	0002	0072	Message not found
0008	0017	0070	The function has already been opened
0008	0023	0001	Invalid Correlator
0008	0023	0065	Correlator has been inactivated due to Host Session Recovery
0008	0047	0146	No received command outstanding

Figure 13 (Part 1 of 2). SPCF API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0049	0009	Storage Not Available
0008	0051	0095	Requests still queued
0008	0065	0070	The function has not been opened
0008	0076	0098	Receive a RUNCMD message (X'0303') call was issued, however no RUNCMD is in the Queue. Issue Receive a command (X'0309') call.
0008	0098	0096	Host session not available
0008	0148	0002	Message or command outstanding
0008	0148	0146	Too many "Receive." calls outstanding

Figure 13 (Part 2 of 2). SPCF API/CS Return Codes

Open the SPCF API/CS

Purpose: To allow an application to use the SPCF functions of the API/CS to communicate with the host.

Setting Up:

1. Provide memory for an ARB.
2. Store "ARB3" in the ARBID field of the ARB.
3. Store 90[3] in the ARB LENGTH field of the ARB.
4. Store the application name in the TARGET NAME field of the ARB
5. Store 0301H in the REQUEST CODE field of the ARB
6. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVC00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Receive a RUNCMD message

Purpose: To receive a message from an operator or a CLIST.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 0303H in the REQUEST CODE field of the ARB
3. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVC00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.
2. Perform processing appropriate for the message received.

Data fields and ARB displacements returned:

Cmdlgth (41)
Command (42)
Recvcorr (46)

You must document for the NetView operator, the format and content of RUNCMD messages received and response messages sent. NetView/PC, API/CS, and NetView only provide for the transportation of the messages, they do not define message content.

Send a RUNCMD response

Purpose: To send a response to a RUNCMD message.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 0302H in the REQUEST CODE field of the ARB
3. If a message is to be sent from a message file, then set up the following ARB fields
   a. MSGTYPE to F (file)
   b. Msgfile with the 4-character name of the message file that contains the message to be sent to NetView
   c. MSGID with the 4-character number of the message in the message file
4. If a message is to be sent from a message buffer, and the message is to be translated then set the following ARB fields:
   a. MSGTYPE to B (buffer)
   b. Msgbuff to the address of the message buffer
   c. Msgcount to the number of messages to be sent from the message buffer
   d. Convert to 'Y'
5. If a message is to be sent from a message buffer, and the message is **NOT** to be translated then set the following ARB fields:
   a. MSGTYPE to B (buffer)
   b. Msgbuff to the address of the message buffer
   c. Msgcount to 1
   d. MBlength to the message length
   e. Convert to 'N'
6. Store the Recvcorr correlator from the received RUNCMD in the SENDCORR field of the ARB.
7. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVC00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

To send unformatted data to the host, you must provide a NetView command processor that can handle the unformatted data (X'1309') major vector key. See Appendix J, "NetView Sample Programs" for command processor source code listings. To use the sample programs to handle unformatted data, add code to "NetView Sample Data Services Command Processor (DSCP)" on page 371, at label MV1309 on page 382, to handle your unique requirements.

To send message IDs and replacement text to NetView, your application must build the NMVT in the form shown in Figure 59 on page 96 and include the X'0A' sub-vector, as shown in Figure 79 on page 104. You must then send the RUNCMD response NMVT as described in "Send a Command Response" on page 43.

RUNCMD Response Message Buffer

The format of the message buffer pointed to by the Msgbuff field of the ARB when CONVERT is 'Y' and MSGTYPE is 'B' is shown in the following table. Several messages in the buffer may be sent. The application sets the Msgcount field of the ARB to the number of messages in the message buffer to be sent. Each message is preceded by a one-byte length field (L) that contains the length of the message. The length of each message must be equal to or less than 253 bytes.

Message data must be in ASCII upper case.

The sum of all the lengths fields, for the number of messages to be sent as specified by the Msgcount field of the ARB, must be equal to or less than 478 minus 2 times Msgcount.

L1	Message Data
L2	Message Data
. . .	. . .
Ln	Message Data

Figure 14. Message buffer format when Convert = 'Y'

Where: Msgcount = n

L1 + L2 ... + Ln =< 478 − (2 X Msgcount)

Send a Message

Purpose: To send an unsolicited message to a NetView Operator from a file or from a buffer.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 030AH in the REQUEST CODE field of the ARB
3. If a message is to be sent from a message file then, set up the following ARB fields
   a. MSGTYPE to F (file)
   b. Msgfile with the 4-character name of the message file that contains the message to be sent to NetView
   c. MSGID with the 4-character number of the message in the message file
4. If a message is to be sent from a message buffer then, set the following ARB fields:
   a. MSGTYPE to B (buffer)
   b. Msgbuff to the address of the message buffer
   c. Msgcount to the number of messages to be sent from the message buffer
   d. Convert to 'Y' (yes)
5. Store the NetView operator's name in the Operator Name field of the ARB.
6. Set the segment and offset register pair (AX−DX) to point to the start of the ARB.

CALL DCJVC00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.

Note: The "Convert" field is ignored. All unsolicited messages to the Host are converted.

Receive a Command

Purpose: Receive an unparsed command NMVT. The application is required to parse the NMVT to determine the command.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 0309H in the REQUEST CODE field of the ARB
3. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVC00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.
2. Perform processing appropriate for the command received.

Data fields and ARB displacements returned:

PARSE ID (7)
Command (42)
Recvcorr (46)

Send a Command Response

Purpose: Send a response to a command from NetView. The response NMVT is sent to NetView as received from the application. The application is required to format the NMVT.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 030BH in the REQUEST CODE field of the ARB
3. Store the Recvcorr correlator from the received command in the SENDCORR field of the ARB.
4. Store the length of the response NMVT in the Putreply length field of the ARB.
5. Store the address of the response NMVT in the Putreply field of the ARB.
6. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVC00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.

Send Error Sense

Purpose: Send error sense data to NetView in response to a command. The sense data may be defined by the application. The X'7D' subvector is used.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 030CH in the REQUEST CODE field of the ARB
3. Store the Recvcorr correlator from the received command in the SENDCORR field of the ARB.
4. Store the appropriate values in the following ARB fields:
   SENSETYPE (See Figure 15)
   LCCSTAT (See Figure 16 on page 45)
   Error Detail (See Figure 17 on page 45)
   User sense
   SV Key
   SF Key
5. Set the segment and offset register pair (AX-DX) to point to the start of the ARB.

CALL DCJVC00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.

Defined SENSETYPE values

Value	Sense Data	Description
0	User sense	A user specified sense code is returned to the requestor. The user sense field in the ARB is used to give the user sense code to the Host and must conform to SNA sense codes.
1	X'084B 0003'	The target manager is not available
2	X'1003 000D'	The request is not accepted or supported by the target.
3	X'081C 0n0m'	The request is accepted by the target, but error(s) occurred during execution. n=LCCSTAT Figure 16 on page 45 and m=Error Detail ARB fields. See Figure 17 on page 45 for defined values.
4	X'086F 0001'	Invalid major vector (MV) length.
5	X'086D 0601'	Required SF (X'01') missing in SV (X'06').
6	X'080C 0006'	Command subvector not recognized.
7	X'086C 3100'	Execute command subvector missing.
8	X'086C 8000'	Test setup data subvector missing.
9	X'0806 0001'	Resource unknown.
10	X'086A svsf'	SF (X'sf') key is invalid for SV (X'sv'). Use ARB fields SV Key and SF Key to show which subfield in which subvector is in error.
11	X'086B svsf'	SF (X'sf') value is invalid for SV (X'sv'). Use ARB fields SV Key and SF Key to show which subfield in which subvector is in error.
12	X'086F sv05'	Subvector (X'sv') length error. Use ARB field SV Key to show which subvector is in error.

Figure 15 (Part 1 of 2). SENSETYPE Values, Data and Descriptions

Value	Sense Data	Description
13	X'086F sf06'	Subfield length error. Use ARB field SF key to show which subvector contains the subfields in error.

Figure 15 (Part 2 of 2). SENSETYPE Values, Data and Descriptions

Defined LCCSTAT Values

Value	Description
1	The link connection component (LCC) and/or the configuration file have recovered from the error. They are in a state prior to the execution of the command.
2	The LCC and/or configuration file are in an unpredictable state.

Figure 16. Defined LCCSTAT Values

Defined Error Detail Values

Value	Description
1	Memory error.
2	File access error.
3	LCCI error.
4	Process error.

Figure 17. Defined Error Detail Values

Close the SPCF API/CS

Purpose: To terminate the use of the SPCF functions of the API/CS. The resources reserved for the application that 'opened' the interface are freed by the SPCF communications functions.

The API/CS can be forced closed (Force='Y') to cause error sense to be sent to the host for all outstanding SPCF commands or messages.

Setting Up:

1. Check that this API/CS ARB has been opened successfully.
2. Store 0304H in the REQUEST CODE field of the ARB
3. If you want to force close the SPCF API/CS, store 'Y' in the Force field of the ARB
4. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVC00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Chapter 6. SPCF Build and Parse

This subroutine is used to parse the NetView™ Release 2 commands:

- LINKDATA
- LINKPD
- LINKTEST
- RUNCMD

and to build responses to the NetView™ Release 2 commands:

- LINKDATA
- LINKPD
- LINKTEST

Note that this subroutine does not build a NMVT for the RUNCMD response message.

The NetView commands are described in *NetView Operation*, SC30-3364. The API/CS supported commands are also described in "API/CS Supported NetView Commands" on page 89. The subroutine performs the functions:

- Parse

  Parse a received SPCF NMVT and provide pointers to the NMVT data in the returned ARB .

- Build

  Build a Response to an SPCF Link command using data pointed to by fields in the ARB or stored in fields in the ARB.

The Link commands supported are LINKPD(8062 major vector key), LINKDATA(8063 major vector key) and LINKTEST(8064 major vector key). The subroutine will function with the SPCF interface (ARBID = ARB3) open or closed. See "API/CS Supported NetView Commands" on page 89 for a description of the supported commands.

The Build and Parse subroutine is used to:

1. Parse NMVTs returned by API/CS "Receive a command" (0309H) requests
2. Build a response NMVT that will be sent by a "Send a command response" (030BH) request.

Parse

Parse SPCF Command ARB

The format of the Parse SPCF Command ARB, and a description of the ARB fields follows:

Disp	Lgth	Name	Description
0	04	ARBID	A 4-character constant that is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump. The 4-character constant 'ARB6' must be stored in the ARBID field.
4	02	REQUEST CODE	A word (2-byte Intel Word (W)) request identifier. Must be X'0000' for Build and Parse.
6	01	ARB LENGTH	The length (36) of the ARB for this API/CS function. The length must be stored into the ARB by the application.
7	01	PARSE ID	A 1-byte field returned by the API that contains the least significant byte of the major vector (MV) key of the command NMVT. The Link commands supported by the Build and Parse subroutines are RUNCMD(X'8061' major vector key), LINKPD(X'8062' major vector key), LINKDATA(X'8063' major vector key), and LINKTEST(X'8064' major vector key). The values returned in this field are X'61', X'62', X'63', and X'64' respectively.
8	01	Reserved	Reserved and must be initialized to binary zeros.
9	02	Return Code	An indicator of the degree of success in performing the request.
11	02	Class	The error class.
13	02	Type	The error type.
15	04	PARSE NMVT	A 4-byte (word offset and word segment) address pointing to a buffer which contains the request NMVT which the user wants parsed. The NMVT in this buffer must be in the same format as if received using Receive a Command (X'0309'). This means the major vector length is in Host format and all text fields are in EBCDIC.
19	01	NUMBER OF NAMES	A 1-byte field containing a count of the number of Resource Names which were found in the parsed NMVT. Each of the three link commands contains a list of resource names destined for the target application. This field will contain the number of names in this list. If the parsed NMVT does not contain a names list this field is set to 00H.
20	04	NAMES	A 4-byte (word offset and word segment) address pointing to a data structure which contains the Resource Names List from the parsed NMVT. The names list is structured beginning with a 1-byte length field followed by a string of ASCII characters whose length is equal to the count in the length field. If there is more than one name in the list the format is repeated with the length byte of the second name directly following the first name. Note that the length byte reflects the actual number of characters in the name and does not account for itself. If the parsed NMVT does not contain a names list then this pointer is set to zero. A layout of the names list data structure is shown in Figure 22 on page 51.
24	02	TEST COUNT	A 2-byte Intel Word (W) field containing the Self Test Count which was obtained in a LINKTEST Command. This field is in 2-byte Intel Word (W) format. If the parsed command is not a LINKTEST request then this field is set to 0000H.
26	01	TEST TYPE	A 1-byte field containing a codepoint which identifies the type of test requested in a LINKTEST Command. Only one codepoint has been defined in the IBM Host supported SPCF LINKTEST Command. This is 01H and indicates a self test has been requested. If the parsed command is not LINKTEST then this field is set to 00H.
27	04	PARSE SENSE DATA	A 4-byte field containing the SNA Error Sense Data which should be returned to the Host if a parse error has been found. Sense codes which can be generated by the parse subroutine are shown in Figure 21 on page 50. If no parse error is found this field will be set to 00000000H.

Figure 18 (Part 1 of 2). Parse SPCF Command ARB

Disp	Lgth	Name	Description
31	01	COMMAND LENGTH	A 1-byte field containing the length of the command text resulting from parsing a RUNCMD. If the parsed command is not RUNCMD then this field is set to 00H.
32	04	COMMAND	A 4-byte (word offset and word segment) address pointing to a data buffer which contains the parsed command text from a RUNCMD. The parsed command text will be in ASCII format. If the parsed command is not a Run then this pointer will be set to zero.

Figure 18 (Part 2 of 2). Parse SPCF Command ARB

Parse SPCF Command API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0004	0000	0000	Parse error, see Parse sense data
0008	0001	0047	Invalid request
0008	0004	0131	Major vector unknown, can not parse

Figure 19. Parse SPCF Command API/CS Return Codes

Parse Request

Purpose: To parse a received SPCF NMVT and provide pointers to the NMVT data in the returned ARB . The PARSE ID field of the ARB is the least significant byte of the NMVT Major Vector (MV) key.

Setting Up:

1. Construct an ARB with ARB LENGTH set to 36
2. Store 'ARB6' in the ARBID field of the ARB
3. Store 0000H in the REQUEST CODE field of the ARB
4. Store the address of the NMVT to be parsed in the PARSE NMVT field of the ARB.
5. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVB00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.
2. Check the PARSE SENSE DATA. If non-zero, code the application to take action appropriate for the sense returned. See Figure 21 on page 50 for an explanation of possible returned sense.

   To return these codes to the host, store the sense data in the 'User sense' field of the SPCF ARB and call the API/CS with the 'Send Error Sense' (030CH) request code.

Returned ARB Data Fields

ARB field	Disp	LINKDATA	LINKPD	LINKTEST	RUNCMD	OTHER
PARSE ID	7	63H	62H	64H	61H	nnH
NUMBER OF NAMES	18	XX	XX	XX		
NAMES	19	XX	XX	XX		
TEST COUNT	23			XX		
TEST TYPE	25			XX		
PARSE SENSE DATA	27	XX	XX	XX	XX	XX
COMMAND LENGTH	30				XX	
COMMAND	31				XX	

Figure 20. Parse Data fields and ARB Displacements Returned

Parse Sense Data Definitions

The possible parse sense codes that can be returned by the Parse utility are shown in the following table.

Sense Data	Description
X'086C 3100'	A RUN command was parsed however the RUN command subvector is missing.
X'086F 3105'	A RUN command was parsed however the RUN command subvector has an incorrect length.
X'086D 0601'	For either a LINKPD, LINKDATA, OR LINKTEST command the subfield containing the resource names list is missing.
X'086F 0606'	For either a LINKPD, LINKDATA, OR LINKTEST command the subfield containing the resource names list has a length error.
X'086B 0601'	For either a LINKPD, LINKDATA, OR LINKTEST command the subfield containing the resource names list is invalid.
X'086C 8000'	A LINKTEST command was parsed however the test set up subvector is missing.
X'086F 8005'	A LINKTEST command was parsed however the test set up subvector has an invalid length.
X'080C 0006'	A LINKTEST command was parsed however the test command type is unknown.
X'086F 8006'	A LINKTEST command was parsed however the test command type subfield has an invalid length.
X'1003 000D'	The major vector key of the received SPCF command was not recognized by the Parse subroutine. Function not supported. The least significant byte of the major vector (MV) key of the command NMVT is returned in the 1-byte PARSE ID field of the ARB by the API/CS.

Figure 21. PARSE Sense Data and Description

Names List Format

Names List Format:

The parsed Names List Format is shown in the following table.

Note: The Name Length value equals the number of characters in the name field.

```
1-byte     Name        1-byte     Name
Length     Field       Length     Field

┌─────────┬──────────┬─────────┬──────┐
│ Length 1│  Name 1  │ Length N│Name N│
└─────────┴──────────┴─────────┴──────┘

Length of  Name 1 in   Length of  Next
Name 1     ASCII chars Next Name  Name
```

Figure 22. Format for Names List

Build

Build SPCF Reply ARB

The format of the Build SPCF Reply ARB, and a description of the ARB fields follows:

Disp	Lgth	Name	Description
0	04	ARBID	A 4-character constant that is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump. The 4-character constant 'ARB5' must be stored in the ARBID field.
4	02	REQUEST CODE	A word (2-byte Intel Word (W)) request identifier. Must be X'0000' for Build and Parse.
6	01	ARB LENGTH	The length (37) of the ARB for this API/CS function. The length must be stored into the ARB by the application.
7	01	BUILD ID	A 1-byte field used to indicate the ID for the type of SPCF response the Build is being requested to build. Three id's are supported. These are 62H for building a LINKPD response, 63H for building a LINKDATA response, and 64H for building a LINKTEST response.
8	01	Reserved	Reserved and must be initialized to binary zeros.
9	02	Return Code	An indicator of the degree of success in performing the request.
11	02	Class	The error class.
13	02	Type	The error type.
15	04	BUILT NMVT	A 4-byte (word offset and word segment) address pointing to a buffer which contains the response NMVT which has been built as a result of this ARB build request. The NMVT in this buffer is in Host format meaning that any 2-byte Intel Word (W) fields have their bytes reversed and all text fields are in EBCDIC. This pointer is returned to the application program after a successful build. If an error is found while processing data, this field is set to zeroes.
19	02	BUILT NMVT LENGTH	A two-byte field indicating the length of the NMVT which has been built as a result of this ARB build request. This field is in 2-byte Intel Word (W) format. This field is returned to the application program after a successful build. If error is found while processing data, this field is set to zeroes.
21	04	PATH LIST INFO	A 4-byte (word offset and word segment) address pointing to a data structure which defines the Path information that is to be included in the SPCF response. The format of the path information varies for the different Build response IDs. See Figure 25 on page 54.

Figure 23 (Part 1 of 2). Build SPCF Reply ARB

Disp	Lgth	Name	Description
25	01	LINK STATUS	A 1-byte field containing the codepoint which will be used to describe the Link Status in building a response for the LINKPD SPCF Command (Build ID = 62H). This field is ignored for Build IDs of 63H or 64H. The values supported by the Host for this field are 00H through 05H. See "Link Status Value Definitions" on page 54.
26	01	NUMBER OF PROBABLE CAUSES	A 1-byte field containing the number of Probable Cause codepoints which are to be included in building a response for the LINKPD SPCF Command (Build ID = 62H). This field is ignored for Build IDs other than 62H. The maximum number of Probable Cause codepoints that can be specified is 124. The meanings of the Probable Cause codepoints are given in SNA Reference Summary GA27-3136. The application program provides this information when requesting the build.
27	04	PROBABLE CAUSE	A 4-byte (word offset and word segment) address pointing to a data area containing the Probable Cause responses to be included in building a response for the LINK PD SPCF Command (Build ID = 62H). This field is ignored for Build IDs other than 62H. The format of the Probable Cause data is shown in Figure 29 on page 56. The application program provides this pointer when requesting the build. See Figure 29 on page 56.
31	01	LINK TEST RESULTS	A 1-byte field containing a codepoint which describes the results of the LINKTEST Command. Three codepoints are supported by the Host. They are 00H for Passed, 01H for Failed, and 02H for Indeterminate. It is used to build subfield X'01' in Figure 75 on page 101. This field is ignored for Build IDs other than 64H.
32	01	TEST TYPE	A 1-byte field containing a codepoint which describes the type of test performed on the link. Two codepoints are supported by the Host. They are 00H for Background Self Test executed, and 01H for Self Test executed when requested. It is used to build subfield X'02' in Figure 75 on page 101. This field is ignored for Build IDs other than 64H.
33	02	TEST COUNT REQUESTED	A 2-byte Intel Word (W) field indicating the Test Count received in the LINKTEST request. It is used to build subfield X'03' in Figure 75 on page 101. This field is ignored for Build IDs other than 64H.
35	02	TEST COUNT EXECUTED	A 2-byte Intel Word (W) field indicating the number of times the test was actually executed. It is used to build subfield X'04' in Figure 75 on page 101. This field is ignored for Build IDs other than 64H.

Figure 23 (Part 2 of 2). Build SPCF Reply ARB

Build SPCF Reply API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0008	0001	0019	Invalid NMVT length
0008	0001	0047	Invalid request
0008	0001	0076	Invalid Build ID
0008	0001	0085	Invalid number of probable causes
0008	0001	0111	Invalid value for Link Connection Component(LCC) data

Figure 24 (Part 1 of 2). Build SPCF Reply API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0001	0114	Data conversion failed (ASCII to EBCDIC)
0008	0002	0015	Path not found
0008	0019	0057	Length error in resource type or name
0008	0019	0092	Length error in LCC data value or name

Figure 24 (Part 2 of 2). Build SPCF Reply API/CS Return Codes

Build Request

Purpose: To build a Response NMVT to an SPCF Link command using data pointed to by fields in the ARB or stored in fields in the ARB. The NMVT Major Vector (MV) key and NMVT format is determined by the code store in the BUILD ID field of the ARB.

Setting Up:

1. Construct an ARB with ARB LENGTH set to 37
2. Store 'ARB5' in the ARBID field of the ARB
3. Store 0000H in the REQUEST CODE field of the ARB
4. Store the code for the supported build function required in the BUILD ID field of the ARB
   a. 62H for LINKPD
   b. 63H for LINKDATA
   c. 64H for LINKTEST
5. Store the address of the path information in the PATH LIST INFO field of the ARB.
6. If BUILD ID is set to 62H, then set the following fields in the ARB:
   a. Link status
   b. Number of probable causes
   c. Probable cause
7. If BUILD ID is set to 64H, then set the following fields in the ARB:
   a. Link test results
   b. Test type
   c. Test count requested
   d. Test count executed
8. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVB00

On Return:

1. Check AX and DX registers and the RC. Code the application to take action appropriate for each error RC.

Data fields and ARB displacements returned:

BUILT NMVT (15)
BUILT NMVT LENGTH (19)

Link Status Value Definitions

The appropriate Link Status value is stored in the LINK STATUS field of the Build SPCF Reply ARB. It is used by the Build subroutine to build the X'82' subvector. See Figure 64 on page 97.

00H No failure detected, resource name and type and probable cause information parameters are not present

01H Detected failure, failing resource isolated; resource name and type has a single element identifying the failing LCC, probable cause information is present

02H Detected failure, failing resource not isolated; resource name and type identifies the segment where the failure might have occurred, probable cause information is present

03H Detected failure, failing resource is on the link connection, outside the scope of the Link Connection Subsystem Manager (LCSM), and upstream from the link segment (i.e., toward the Using Node). resource name and type identifies the segment that is downstream of the detected failure, probable cause information is present

04H Detected failure, failing resource is on the link connection, outside of the scope of the LCSM, and inside the link segment identified; resource name and type identifies the segment, probable cause information is present

05H Detected failure, failing resource is on the link connection, outside of the scope of the LCSM, and downstream from the link segment identified; resource name and type identifies the segment that is upstream of the detected failure, probable cause information is present

Path Information List Control Blocks

Disp	Lgth	Name	Description
0	02	LCC Number	The number of LCC resources in the path, in 2-byte Intel Word (W) format.
2	04	LCC PTR	A pointer to the first LCC Description data structure. See Figure 26 for LINKPD response. See Figure 27 on page 55 for LINKDATA or LINKTEST response.

Figure 25. Path Configuration Information CB

LINKPD LCC Description Control Block

Disp	Lgth	Name	Description
0	01	Resource Type Length	The length of the Hierarchy Resource Type field. Valid lengths are between 1 and 8.
1	08	Resource Type	An 8-character field containing the Hierarchy Resource Type, in ASCII.
9	01	Resource name length	The length of the Hierarchy Resource Name field. Valid lengths are between 1 and 8.
10	08	Resource Name	An 8-character field containing the hierarchy resource name, in ASCII.

Figure 26. LINKPD LCC Description CB

This data structure is used by the Build subroutine to construct a X'1307' major vector and the X'05' subvectors in the X'1307' major vector, as shown in Figure 64 on page 97.

LCC description data, as shown in Figure 26, must be repeated in sequential storage for each resource that has information returned. One data structure is required for each resource in the path, and they must be in downstream order.

LINKDATA And LINKTEST LCC Description Control Block

Disp	Lgth	Name	Description
0	01	Resource Type Length	The length of the Hierarchy Resource Type field. Valid lengths are between 1 and 8.
1	08	Resource Type	An 8-character field containing the Hierarchy Resource Type, in ASCII.
9	01	Resource name length	The length of the Hierarchy Resource Name field. Valid lengths are between 1 and 8.
10	08	Resource Name	An 8-character field containing the Hierarchy Resource Name, in ASCII.
18	02	LCC Data Number	A 2-byte Intel Word (W) formatted field containing the number of data elements related to this resource that will be returned.
20	04	LCC Data PTR	A pointer to the first LCC data element. See Figure 28.

Figure 27. LINKDATA and LINKTEST LCC Description CB

This data structure is used by the Build subroutine to construct a X'1307' major vector and the X'05' and X'80' subvectors in the X'1307' major vector, as shown in Figure 69 on page 98.

LCC description data, as shown in Figure 27, must be repeated in sequential storage for each resource that has information returned. One data structure is required for each resource in the path, and they must be in downstream order.

LINKDATA And LINKTEST LCC Data Control Block

Disp	Lgth	Name	Description
0	01	LCC Data Value Type	Indicator of how this LCC Data element will be displayed at the focal point. Valid values are: 02H = HEXADECIMAL VALUE; 03H = CHARACTER VALUE; 04H = DECIMAL VALUE; 05H = BIT STRING VALUE
1	01	LCC Data Value Length	The length of the LCC data value in bytes. Bit string lengths should also be in number of bytes. Valid lengths are 1 to 255.
2	01	Reserved	Reserved
3	04	LCC Data Value PTR	A 4-byte (word offset and word segment) address pointer to the actual LCC data.
7	01	LCC Data Name Length	The length of the LCC data name. Valid lengths are 1 to 255.
8	*	LCC Data Name	The LCC Data Name in upper case ASCII.

Figure 28. LINKDATA and LINKTEST LCC Data CB

This data structure is used by the Build subroutine to construct a X'80' subvector. If the reply is to a LINKDATA (X'63' in the BUILD ID field of the ARB) see Figure 70 on page 99. If the reply is to a LINKTEST (X'64' in the BUILD ID field of the ARB) see Figure 76 on page 102. The X'80' subvector is included in the X'1307' major vector, as shown in Figure 69 on page 98 or Figure 74 on page 100.

LCC Data, as shown in Figure 28, must be repeated in sequential storage for each resource in the path. The data structures must be in downstream order.

LCC Data

If decimal value, the data should be in 4-byte Intel Double Word (DW) format. If character, the data should be in ASCII format.

Probable Cause

2-byte prob. cause	2-byte prob. cause	2-byte prob. cause	. . .

Figure 29. Format for Probable Cause Data

Probable cause data can be repeated up to 124 times. This data is used to construct the probable cause subvector X'93', as shown in Figure 66 on page 98 of the LINKPD response NMVT, as shown in Figure 64 on page 97. The probable cause data is pointed to by the PROBABLE CAUSE field of the Build SPCF Reply ARB.

Chapter 7. Host Data Facility Subroutine Calls

The API/CS provides for the transfer (send or receive) of DOS files with a Host CICS application[4]. The request codes used by the Host Data Facility API/CS and descriptions are:

0401H	Open the Host Data Facility API/CS
0402H	Send file data
0403H	Receive file data
0405H	Check the status of the request
0406H	Stop file data transfer
0404H	Close the Host Data Facility API/CS

One file data transfer (send or receive) of a DOS file may be in progress for an application at any time.

To use the API/CS to send DOS file data to the host or to receive DOS file data from the host, code the following API/CS calls:

1. Call DCJVD00 with request code 0401H to open the Host Data Facility API/CS.
2. Call DCJVD00 with request code 0402H to send file data to the host
3. Call DCJVD00 with request code 0403H to start the receipt of file data from the host CICS application.
4. Call DCJVD00 with request code 0405H to check the status of the request. Calls to Check the status of the request should be made at 1-minute intervals until the file has been completely sent or received.
5. Call DCJVD00 with request code 0406H to stop the file data transfer.
6. Call DCJVD00 with request code 0404H to close the Host Data Facility API/CS when there are no more files to send or receive.

Although only one DOS file data transfer may be in progress for an application at any time, an application may transfer many DOS files with a single open. Each transfer must be complete before another may be started.

Host Data Facility ARB

The format of the Host Data Facility ARB, and a description of the ARB fields follows:

Disp	Lgth	Name	Description
0	04	ARBID	A 4-character constant that is used by the API/CS to verify the start of the ARB and serves as an 'eye catcher' in a storage dump. The 4-character constant 'ARB4' must be stored in the ARBID field.
4	02	REQUEST CODE	A word (2-byte Intel Word (W)) request identifier. Each request has a unique code that must be stored in the ARB by the Application. The first byte identifies the function and the second byte identifies the request.
6	01	ARB LENGTH	The length (45) of the ARB for this API/CS function. The length must be stored into the ARB by the application.

Figure 30 (Part 1 of 2). Host Data Facility ARB

[4] To transfer NetView/PC files, the Host Data Facility facility should be selected from the operator service panel.

Disp	Lgth	Name	Description
7	02	Reserved	Reserved and must be initialized to binary zeros.
9	02	Return Code	An indicator of the degree of success in performing the request.
11	02	Class	The error class.
13	02	Type	The error type.
15	04	PCFILE	A 4-byte (word offset and word segment) address pointing to a buffer that contains the fully qualified file name (path, filename and the extension, if used) as defined by DOS, of the file to be sent to, or received from, the host CICS sub-system. (Mandatory)
19	01	PCFLGTH	A 1-byte field containing the number of characters (1 to 31) of the file name in the buffer pointed to by "PCFILE"
20	04	HOSTFILE	A 4-byte (word offset and word segment) address pointing to a buffer containing the 1 to 8 character entry name in the CICS file name table. (Mandatory)
24	01	HFLGTH	A 1-byte field containing the number of characters (1 to 8) of the name in the buffer pointed to by "HOSTFILE"
25	04	Start byte	A 32-bit integer. The offset to the first byte within a file to be transmitted to the host. (Optional, defaulted to start of file.)
29	01	xpc	(N|T) Character (1) keyword that indicates whether the file is to be transmitted in a transparent (T) or a non-transparent (N) mode. Defaults to 'N' for anything except 'T'.
30	02	blkz	The length of data blocks to be sent to the host. The range is from 512 to 3750 bytes. (Defaulted to 3750 bytes)
32	08	Reserved	Reserved
40	04	Nextbyte	A 32-bit integer. The offset to the next byte to be transmitted within a file. Returned on STOP requests made while transferring files from NetView/PC to the host. May be used to restart transfers that are stopped.
44	01	HDFState	A 1-byte field that is returned on STOP requests made while files are being transferred. X'00' = Transfer is in progress (not stopped). X'40' = Transfer has been stopped abnormally or when STOP is requested by the application. X'80' = Transfer has completed.

Figure 30 (Part 2 of 2). Host Data Facility ARB

Host Data Facility API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0004	0005	0114	File Transfer Program busy transferring files
0004	0098	0009	Storage reduced - BLKZ reduced
0008	0001	0010	BLKZ invalid
0008	0001	0014	File password invalid
0008	0001	0021	Invalid catalog record
0008	0001	0024	File write access locked
0008	0001	0026	Invalid record (journal)
0008	0001	0047	Invalid request

Figure 31 (Part 1 of 3). Host Data Facility API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0001	0061	OFFSET invalid
0008	0001	0068	Invalid data in file
0008	0001	0075	CICS attributes invalid
0008	0001	0076	Invalid Build ID
0008	0001	0083	Hostname invalid
0008	0001	0085	Invalid number of probable causes
0008	0001	0109	Invalid filespec
0008	0001	0111	Invalid value for Link Connection Component(LCC) data
0008	0001	0114	Data conversion failed (ASCII to EBCDIC)
0008	0001	0115	Error detected at the host
0008	0001	0142	Invalid stop, not authorized
0008	0002	0015	Path not found
0008	0002	0068	File not found
0008	0002	0085	Status not found
0008	0002	0115	Host file not found
0008	0004	0115	Host file is full
0008	0004	0131	Major vector unknown, can not parse
0008	0005	0115	Host file is in use
0008	0008	0115	File damaged at host
0008	0009	0115	Host file space is unavailable
0008	0013	0004	PC disk is full
0008	0013	0008	Physical disk error
0008	0013	0130	Disk drive not ready
0008	0017	0070	The function has already been opened
0008	0019	0057	Length error in resource type or name
0008	0019	0092	Length error in LCC data value or name
0008	0022	0068	File non-shared and open
0008	0039	0114	Timeout - No reply from host
0008	0050	0068	Too many open files
0008	0053	0008	Unrecoverable DOS error
0008	0053	0015	Invalid disk drive specified
0008	0065	0070	The function has not been opened
0008	0070	0115	Host file not open
0008	0076	0082	Multiple replies requested when link status equals zero
0008	0082	0024	Translation denied for this file
0008	0082	0114	Invalid stop — No transfer in progress
0008	0082	0115	Not authorized to transfer host file
0008	0083	0115	Invalid host file name
0008	0093	0115	Invalid CICS code point received
0008	0094	0113	APPC — Abend
0008	0094	0115	Host aborted file transfer
0008	0098	0009	Storage not available

Figure 31 (Part 2 of 3). Host Data Facility API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0098	0068	Security file not available
0008	0098	0113	APPC not available
0008	0098	0114	Host data transfer program not available
0008	0098	0115	Host file temporarily not available
0008	0110	0067	Filename reserved to DOS
0008	0118	0069	CICS Security failure
0008	0118	0083	Incorrect partner LU name
0008	0118	0096	CICS allocation failure
0008	0118	0115	CICS session failure – No retry
0008	0123	0096	Invalid response, system
0008	0123	0115	Invalid message from host

Figure 31 (Part 3 of 3). Host Data Facility API/CS Return Codes

Open the Host Data Facility API/CS

Purpose: To allow an application to use the Host Data Facility functions of the API/CS to transfer DOS file data to the host.

Setting Up:

1. Provide memory for an ARB
2. Store "ARB4" in the ARBID field of the ARB.
3. Store request code 0401H in the request code field of the ARB.
4. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVD00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Send File Data

Purpose: To send DOS file data to the host.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Store the required fields in the Host Data Facility ARB
3. Store request code 0402H in the request code field of the ARB.
4. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVD00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Receive File Data

Purpose: To receive DOS file data from the host.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Store the required data fields in the Host Data Facility ARB
3. Store request code 0403H in the request code field of the ARB.
4. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVD00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Check the Status of a Host Data Facility Request

Purpose: To determine the status of a request to transfer data.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Store request code 0405H in the request code field of the ARB.
3. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVD00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Stop File Data Transfer

Purpose: To stop the transfer of file data to or from the host.

Setting Up:

1. Check that the API/CS has been opened successfully.
2. Store request code 0406H in the request code field of the ARB.
3. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVD00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Close the Host Data Facility API/CS

Purpose: To terminate the use of the Host Data Facility functions. The resources reserved for the application that 'opened' the interface are freed.

Setting Up:

1. Store request code 0404H in the request code field of the ARB.
2. Set the segment and offset register pair (AX–DX) to point to the start of the ARB.

CALL DCJVD00

On Return: Check AX and DX registers and the RC. Code the application to take action appropriate for each RC.

Part 3. Reference Information

Appendix A. API/CS Reference Information 67
Return Code List 67
List of all API/CS Return Codes 67
DOS Error Codes 71
Translation of NMVT Data Fields 72
Naming Conventions 73
NetView/PC Prefix 73
Panel Name 74
Panel Field Name 74
NetView/PC Message Format 75

Appendix B. Alert Major Vector Formats 77
Non-generic Alert Format 77
Tables of Text for X'91' Subvector Support 78
Alert Type 78
General Cause Table 79
Specific Component Table 79
NetView/PC X'9F' Subvector 80
NetView/PC Cause Subfields 82
NetView/PC ALERT SV X'9F' Code Point File : DUPALGTF.TXT 83
ALERT Description Records 84
Generic Alert Format 87

Appendix C. Service Point Command Data 89
API/CS Supported NetView Commands 89
LINKDATA 89
LINKPD 91
LINKTEST 92
RUNCMD 93
Service Point Command vectors 94
NMVT Length Algorithms 94
NMVT Header 95
Service Point Command Major Vectors 95
RUNCMD Vectors 95
Unsolicited Operator Message Vectors 96
LINKPD Vectors 97
LINKDATA Vectors 98
LINKTEST Vectors 99
Common Subvectors 102

Appendix D. Suggested Command Formats 107
Suggested Physical Device Management Commands 107
LINK—CHANGE 107
Purpose of Command 107
Actions Taken by Receiver 107
Inputs: 107
Outputs: 107
LINK—DISPLAY 108
Purpose of Command 108
Actions Taken by Receiver 108
Inputs: 108
Outputs: 108

Configuration Data Base Management Commands 108
RESOURCE—DISPLAY 108
Purpose of Command 109
Actions Taken by Receiver 109
Inputs: 109
Outputs: 109
RESOURCE—CHANGE 109
Purpose of Command 109
Actions Taken by Receiver 109
Inputs: 110
Outputs: 110
PATH—DISPLAY 110
Purpose of Command 110
Actions Taken by Receiver 110
Inputs: 110
Outputs: 110
PATH—CHANGE 111
Purpose of Command 111
Actions Taken by Receiver 111
Inputs: 111
Outputs: 111

Appendix E. Panel Development Rules 113
Applicability and Conformance 113
Requirements 113
Panel Design 113
Types of Panel Elements 114
Common Panel Elements 115
Panel Body Elements 120
Panel Types 127
Menu Panels 127
Entry Panels 129
List Panels 133
Information Panels 134
Mixing Panel Types 135
Mixed Panel Examples and Guidance 136
Panel Dialog Management 137
Introduction 137
Panel Interaction Techniques 137
Dialog Control 138
Dialog Control Objects 139
Dialog Control Actions 139
Dialog Control Techniques 142
Advanced Dialog Control Techniques 143
Scrolling 143
Function Key Utilization 144
Dedicated Function Keys 145
Programmable Function Keys 145
Required Function Keys 146
Messages and Prompts 146
Message/Prompt Presentation 146
Message/Prompt Format 146
Message Types 147
Message Rules 147
Message and Prompt Text Guidelines 148
Help Facility 148

Displaying Help Panels 149
Common Help Panel Elements 149
Help Interaction Techniques 150
Types of Help 150
Content of Help Panels 151
Color and Emphasis 151
Overview 151
Classes of Data 152
File Management Techniques Using the List Panel 155
Maintenance Menu Panel 158
List Panel 158
Delete Function 158
List Panel Dialog Control 159
Change Function 159
Change Panel Dialog Control 159
Add Function 160
Add Panel Dialog Control 160

Appendix A. API/CS Reference Information

Return Code List

List of all API/CS Return Codes

The meaning of the return code, class, and type combinations is described in the following table:

Return Code	Class Field	Type Field	Description
0000	0000	0000	Request processed without error
0002	0000	0000	SPCF Request Queue is empty
0004	0000	0000	Parse error, see Parse sense data
0004	0005	0114	File Transfer Program busy transferring files
0004	0098	0009	Storage reduced - BLKZ reduced
0008	0001	0010	BLKZ invalid
0008	0001	0014	File password invalid
0008	0001	0019	Invalid NMVT length
0008	0001	0021	Invalid catalog record
0008	0001	0023	Invalid NMVT key field
0008	0001	0024	File write access locked
0008	0001	0026	Invalid record (journal)
0008	0001	0040	Date/Time subvector data invalid
0008	0001	0041	Basic subvector data invalid
0008	0001	0042	PSID subvector data invalid
0008	0001	0043	Hierarchy Names subvector data invalid
0008	0001	0044	NetView/PC Alert subvector data invalid
0008	0001	0045	Text subvector data invalid
0008	0001	0047	Invalid request
0008	0001	0061	OFFSET invalid
0008	0001	0068	Invalid data in file
0008	0001	0072	Invalid MSGTYPE
0008	0001	0075	CICS attributes invalid
0008	0001	0076	Invalid Build ID
0008	0001	0083	Hostname invalid
0008	0001	0085	Invalid number of probable causes
0008	0001	0109	Invalid filespec
0008	0001	0111	Invalid value for Link Connection Component(LCC) data
0008	0001	0114	Data conversion failed (ASCII to EBCDIC)
0008	0001	0115	Error detected at the host
0008	0001	0136	Invalid character for ASCII to EBCDIC translation

Figure 32 (Part 1 of 5). List of all API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0001	0142	Invalid stop, not authorized
0008	0001	0144	Detail qualifier subvector data invalid
0008	0001	0147	LAN subvector data invalid
0008	0002	0009	Storage not available
0008	0002	0015	Path not found
0008	0002	0040	Date/Time subvector missing
0008	0002	0041	Basic subvector missing
0008	0002	0042	PSID subvector missing
0008	0002	0043	Hierarchy Names subvector missing
0008	0002	0044	NetView/PC Alert subvector missing
0008	0002	0066	The requested function is not available
0008	0002	0068	File not found
0008	0002	0072	Message not found
0008	0002	0085	Status not found
0008	0002	0115	Host file not found
0008	0002	0160	Hierarchy resource list subvector missing
0008	0002	0162	Link station data subvector missing
0008	0002	0163	Generic alert data subvector missing
0008	0002	0164	Probable cause subvector missing
0008	0002	0165	User cause subvector missing
0008	0002	0166	Install cause subvector missing
0008	0002	0167	Failure cause subvector missing
0008	0002	0168	Undetermined cause subvector missing
0008	0002	0169	Detailed data subvector missing
0008	0002	0170	Self-defining text message subvector missing
0008	0003	0040	Duplicate Date/Time subvector
0008	0003	0041	Duplicate Basic subvector
0008	0003	0042	Duplicate PSID subvector
0008	0003	0043	Duplicate Hierarchy Names subvector
0008	0003	0044	Duplicate NetView/PC Alert subvector
0008	0003	0045	Duplicate Text subvector
0008	0003	0070	The function has already been opened
0008	0003	0144	Duplicate Detail Qualifier subvector
0008	0003	0147	Duplicate LAN subvector
0008	0003	0160	Duplicate Hierarchy resource list subvector
0008	0003	0162	Duplicate Link station data subvector
0008	0003	0163	Duplicate Generic alert data subvector
0008	0003	0164	Duplicate Probable cause subvector
0008	0003	0165	Duplicate User cause subvector
0008	0003	0166	Duplicate Install cause subvector
0008	0003	0167	Duplicate Failure cause subvector
0008	0003	0168	Duplicate Undetermined cause subvector

Figure 32 (Part 2 of 5). List of all API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0003	0169	Duplicate Detailed data subvector
0008	0003	0170	Duplicate Self-defining text message subvector
0008	0004	0115	Host file is full
0008	0004	0131	Major vector unknown, can not parse
0008	0005	0115	Host file is in use
0008	0008	0008	Unexpected error. See other return codes for furtherexplanation
0008	0008	0023	Major vector key field format error
0008	0008	0040	Date/Time subvector format error
0008	0008	0041	Basic subvector format error
0008	0008	0042	PSID subvector format error
0008	0008	0043	Hierarchy Names subvector format error
0008	0008	0044	NetView/PC Alert subvector format error
0008	0008	0045	Text subvector format error
0008	0008	0096	NetView/PC Alert Manager not available
0008	0008	0115	File damaged at host
0008	0008	0144	Detail Qualifier subvector format error
0008	0008	0147	LAN subvector format error
0008	0008	0160	Hierarchy resource list subvector format error
0008	0008	0162	Link station data subvector format error
0008	0008	0163	Generic alert data subvector format error
0008	0008	0164	Probable cause subvector format error
0008	0008	0165	User cause subvector format error
0008	0008	0166	Install cause subvector format error
0008	0008	0167	Failure cause subvector format error
0008	0008	0168	Undetermined cause subvector format error
0008	0008	0169	Detailed data subvector format error
0008	0008	0170	Self-defining text message subvector format error
0008	0009	0115	Host file space is unavailable
0008	0012	0068	File I/O error
0008	0012	0096	NetView/PC Alert Manager and host session are not available
0008	0013	0004	PC disk is full
0008	0013	0008	Physical disk error
0008	0013	0130	Disk drive not ready
0008	0017	0070	The function has already been opened
0008	0019	0057	Length error in resource type or name
0008	0019	0092	Length error in LCC data value or name
0008	0022	0068	File non-shared and open
0008	0023	0001	Invalid Correlator
0008	0023	0065	Correlator has been inactivated due to Host Session Recovery
0008	0027	0079	The ARB is unused or closed
0008	0039	0114	Timeout - No reply from host
0008	0047	0146	No received command outstanding

Figure 32 (Part 3 of 5). List of all API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0049	0009	Storage Not Available
0008	0050	0068	Too many open files
0008	0050	0115	Checkpoint size too large
0008	0051	0095	Requests still queued
0008	0053	0008	Unrecoverable DOS error
0008	0053	0015	Invalid disk drive specified
0008	0057	0115	Host resource limit reached
0008	0065	0070	The function has not been opened
0008	0065	0078	CP-PU not active; retry
0008	0070	0003	Application name is already open
0008	0070	0115	Host file not open
0008	0076	0082	Multiple replies requested when link status equals zero
0008	0076	0098	Receive a RUNCMD message (X'0303') call was issued, however no RUNCMD is in the Queue. Issue Receive a command (X'0309') call.
0008	0082	0024	Translation denied for this file
0008	0082	0114	Invalid stop — No transfer in progress
0008	0082	0115	Not authorized to transfer host file
0008	0083	0115	Invalid host file name
0008	0093	0115	Invalid CICS code point received
0008	0094	0113	APPC—Abend
0008	0094	0115	Host aborted file transfer
0008	0096	0098	Alert Router is currently not available
0008	0098	0009	Storage not available
0008	0098	0063	System record not available
0008	0098	0068	Security file not available
0008	0098	0096	Host session not available
0008	0098	0113	APPC not available
0008	0098	0114	Host data transfer program not available
0008	0098	0115	Host file temporarily not available
0008	0110	0067	Filename reserved to DOS
0008	0117	0115	Request processed without error for NetView/PC Alert Manager, but did not process for host
0008	0117	0116	Request processed without error for host, but did not process for, or received a warning from, the NetView/PC Alert Manager
0008	0118	0069	CICS Security failure
0008	0118	0083	Incorrect partner LU name
0008	0118	0096	CICS allocation failure
0008	0118	0115	CICS session failure — No retry
0008	0123	0096	Invalid response, system
0008	0123	0115	Invalid message from host
0008	0148	0002	Message or command outstanding
0008	0148	0146	Too many "Receive." calls outstanding
0008	0159	0002	Dependent key missing

Figure 32 (Part 4 of 5). List of all API/CS Return Codes

Return Code	Class Field	Type Field	Description
0008	0159	0040	Date/Time subvector key dependency error
0008	0159	0041	Basic subvector key dependency error
0008	0159	0042	PSID subvector key dependency error
0008	0159	0043	Hierarchy Names subvector key dependency error
0008	0159	0044	NetView/PC Alert subvector key dependency error
0008	0159	0045	Text subvector key dependency error
0008	0159	0144	Detail Qualifier subvector key dependency error
0008	0159	0147	LAN subvector key dependency error
0008	0159	0160	Hierarchy resource list subvector key dependency error
0008	0159	0162	Link station data subvector key dependency error
0008	0159	0163	Generic alert data subvector key dependency error
0008	0159	0164	Probable cause subvector key dependency error
0008	0159	0165	User cause subvector key dependency error
0008	0159	0166	Install cause subvector key dependency error
0008	0159	0167	Failure cause subvector key dependency error
0008	0159	0168	Undetermined cause subvector key dependency error
0008	0159	0169	Detailed data subvector key dependency error
0008	0159	0170	Self-defining text message subvector key dependency error

Figure 32 (Part 5 of 5). List of all API/CS Return Codes

DOS Error Codes

The multi-tasking environment of NetView/PC required that the handling of DOS critical errors be modified. In DOS, critical errors are handled by DOS and the return codes for these particular errors are not normally returned to calling programs.

Under NetView/PC all DOS critical errors are returned to the calling program with the following indications of the problem. Note that there are other error returns, in addition to these, that are documented with the DOS descriptions of the calls.

- The Carry Bit will be set
- The error code will be placed in the AL

The error codes are:

80H or 128 D Attempt to write on write-protected diskette

81H or 129 D Unknown unit

82H or 130 D Drive not ready

83H or 131 D Unknown command

84H or 132 D Data Error (CRC)

85H or 133 D Bad request structure length

86H or 134 D Seek error

87H or 135 D Unknown media type

88H or 136 D Sector not found

89H or 137 D Printer out of paper

8AH or 138 D Write fault

8BH or 139 D Read fault

8CH or 140 D General failure

These codes are the Extended Error Codes (given in the DOS Technical reference manual) in the range 19D to 31D(DOS critical errors). The codes are generated by adding 128D to the critical error code values that range from 0 to 12D, internally in DOS. The consequences of all of this are as follows:

1. Users of the DOS partition, under NetView/PC must be made aware that on DOS critical errors, the carry bit will be set on return from a DOS Function call and the AL register will have a value representing the modified error code.
2. DOS COMMAND.COM may display the wrong message on intervention required but the operation is the same (see "DOS Error Codes" on page 71 and the README file on diskettes).
3. The DOS Partition was intended for user programs designed specifically for a NetView/PC environment. Off-the-shelf programs may not operate correctly.

Translation of NMVT Data Fields

NetView/PC API/CS Programs that build or receive NMVTs should process EBCDIC fields in ASCII. NetView/PC translates NMVT EBCDIC fields of received NMVTs and prior to transmission of NMVTs to the host.

The translate table used to translate data from ASCII to EBCDIC is shown in Figure 33 on page 73. The translate table used to translate data from EBCDIC to ASCII is shown in Figure 34 on page 73. These tables are used to translate all fields of NMVTs that are designated as EBCDIC only fields by the SNA architecture. The tables are the same as these listed in the *3278/79 Emulation Control Program Users Guide* for the 3278/78 Emulation Control Program, 6024134.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	4A	E0	4F	5F	6D
6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	6A	D0	A1	07
8	20	21	22	23	24	15	06	17	28	29	2A	2B	2C	09	0A	1B
9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	E1
A	41	42	43	44	45	46	47	48	49	51	52	53	54	55	56	57
B	58	59	62	63	64	65	66	67	68	69	70	71	72	73	74	75
C	76	77	78	80	8A	8B	8C	8D	8E	8F	90	9A	9B	9C	9D	9E
D	9F	A0	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7
E	B8	B9	BA	BB	BC	BD	BE	BF	CA	CB	CC	CD	CE	CF	DA	DB
F	DC	DD	DE	DF	EA	EB	EC	ED	EE	EF	FA	FB	FC	FD	FE	FF

Figure 33. ASCII to EBCDIC Translation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1	10	11	12	13	9D	85	08	87	18	19	92	8F	1C	1D	1E	1F
2	80	81	82	83	84	0A	17	1B	88	89	8A	8B	8C	05	06	07
3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4	20	A0	A1	A2	A3	A4	A5	A6	A7	A8	5B	2E	3C	28	2B	5D
5	26	A9	AA	AB	AC	AD	AE	AF	B0	B1	21	24	2A	29	3B	5E
6	2D	2F	B2	B3	B4	B5	B6	B7	B8	B9	7C	2C	25	5F	3E	3F
7	BA	BB	BC	BD	BE	BF	C0	C1	C2	60	3A	23	40	27	3D	22
8	C3	61	62	63	64	65	66	67	68	69	C4	C5	C6	C7	C8	C9
9	CA	6A	6B	6C	6D	6E	6F	70	71	72	CB	CC	CD	CE	CF	D0
A	D1	7E	73	74	75	76	77	78	79	7A	D2	D3	D4	D5	D6	D7
B	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7
C	7B	41	42	43	44	45	46	47	48	49	E8	E9	EA	EB	EC	ED
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	EE	EF	F0	F1	F2	F3
E	5C	9F	53	54	55	56	57	58	59	5A	F4	F5	F6	F7	F8	F9
F	30	31	32	33	34	35	36	37	38	39	FA	FB	FC	FD	FE	FF

Figure 34. EBCDIC to ASCII Translation

Naming Conventions

The naming conventions used for NetView/PC components, panels, and messages are included with this document. Users are encouraged to follow them for their own NetView/PC applications, so that naming conventions will be consistent for all NetView/PC products and applications.

NetView/PC Prefix

All NetView/PC modules, procedures, panels and messages begin with a unique prefix. The other four or five characters are descriptive of the function. The name must be seven (7) characters long if defining the name of a main procedure or if the code will be link-edited, otherwise the name is eight (8) characters long.

The name format is as follows:

```
  dddaaxxy        where:

    ddd =   is the prefix.
            The following prefixes are used by NetView/PC.
            To avoid confusion they should not be used by
            other applications.

            DCJ        DUS        DUP
            DUQ


      aa =   is the "function identifiers":
For example:
            AL  is the Alert Manager
            AM  is the Access Method Services Manager
            CS  is the CSSA (Alert Router)
            .
            .
            .
            VA  Is the Vendor API Alert Facility Interface
            VB  Is the Vendor API Build and Parse subroutines
            VC  Is the Vendor API Command Facility Interface
            VD  Is the Vendor API Data Facility Interface
            VO  Is the Vendor API Operator Communication
      xx = is the unique identifier used within the function.
           It will always be '00' for Outer
           Procedures with the range of 00-99.

       y = The additional character for eight (8) digit names.
           It will be 0 unless needed to additionally qualify
           the xx identifier.
```

Panel Name

To assist in standardizing panels and panel field names the following has been developed for identifying functions.

Panel names must be eight (8) characters in the following format.

```
  dddaaPXX        where:

    ddd = Prefix (identified at the start of this section)

     aa = Function identifier.  See "Module (Outer Procedures)
                / Macro / Driver Names" in this section.

      P = indicates this is a panel name.

     XX = A unique two character panel identifier
                 (numeric and/or alphabetic).
```

Panel Field Name

The fields defined in a panel must be identified with an eight (8) character name in the following format.

```
aapxxyyy       where:

    aa = Function identifier

     p = indicates this is a panel field name.

    xx = The panel identifier where this field resides
               For variables common to more than one function
               the 'p' and this identifier (xx) will be
               'GML'.

   yyy = A unique three character field identifier
                (numeric and/or alphabetic)
```

NetView/PC Message Format

All messages to the operator are retrieved from one or more disk files of messages. There are two types of messages:

1. Messages retrieved from disk by EZ-VU for display on the message line of an EZ-VU panel, and
2. Messages retrieved from disk by NetView/PC generic system support for display as dynamic information to the operator (i.e. not on the message line of a panel).

The DOS file specification for NetView/PC message files will use the following format:

```
        dddx.MSG
```

Where:

```
    ddd = is the prefix

      x = A for NetView/PC message files, or
          B for EZ-VU message files

    MSG = the required file extension
```

The format of messages contained in all message files is:

```
     nnnn msgtxt. dddxnnnnt
```

Where:

```
   nnnn = the unique message number within the message file
            (0001-9999)

msgtxt. = the actual text of a message to the operator

   dddx = The DOS filename of the message file containing the
          message text (see previous paragraph on names of
          message files)

      t = Is the message classification, according
          to the following:

          I  Information Message.  Provides the user with
             feedback about the state of the application.
```

Typically used to tell the user that input has been accepted and is or has been processed.

W Warning Message. Calls the user's attention to an to an exception condition that is not necessarily an error.

A Action Message. Used to notify the user that an improper action has taken place or attempted, or that the application has had an exception condition and requires user action. An audible alarm must be associated with this message type.

Appendix B. Alert Major Vector Formats

Non-generic alerts use predefined screens at NetView whereas generic Alerts use code points. A code point is a number that indexes into a table of text strings. The strings of text are displayed on the NetView screen.

Generic Alert code points have been defined that provide the ability to describe error and resource types, causes, and recommended actions.

A Hybrid Alert combines a complete non-generic alert and a complete generic alert in one Network Management Vector Transport (NMVT). Complete means that the required subvectors for each alert type (generic or non-generic) must be met within the NMVT. Date/time and PSID subvectors are required in both Alert types but the requirement is satisfied by a single appearance.

Management Services Alert Major Vectors and subvectors must be built as described in *System Network Architecture Formats*, GA27-3136, (formerly called *System Network Architecture Reference Summary*), except use ASCII instead of EBCDIC for all text fields and use the Intel (PC) format for unsigned 16 bit integer (2-byte Intel Word (W)) and unsigned 32 bit integer (4-byte Intel Double Word (DW)) fields. NetView/PC will convert ASCII to EBCDIC and prepare fixed(16) and fixed(32) fields for the host environment as required.

Non-generic Alert Format

See *NetView Customization*, Chapter 2, Using NMVT Support for User Written Programming, for further information.

NetView/PC recognizes a unique (X'9F') subvector that is sent to the local Alert Manager. The X'9F' subvector is not sent to the host when it is included in an Alert NMVT. The Alert Router strips the X'9F' subvector from the Alert before it is sent to the host.

Required subvectors in all Alert Major Vector NMVTs:

- X'01' - Date/Time - only one allowed per NMVT.
- X'03' - Hierarchy Name List - only one allowed per NMVT.
  1. The reserved byte, byte two should be X'03'.
     When X'03', NetView will only use the resource names and types in this (HNL) subvector.
     When anything else, NetView will concatenate the first two resource names and types in this (HNL) subvector with three VTAM resource names and types (PU name, link, and controller).
  2. Five Maximum resource names and types are allowed in the HNL subvector.
  3. First Hierarchy Names List Entry must contain the resource (NetView/PC or PU) name with the type identifier : 'SP ' (Service Point). The resource name is located in DOS file 'DCJSFSPN.REC'. The file has a single 11-byte record in the following format.
     resource name - Eight (8) bytes
     line control - Three (3) bytes - 0D0A1AH

- X′10′ - PSID - two maximum per NMVT.
- X′11′ - Multiples allowed in each PSID subvector.
  1. First PID must have product classification of software (X′04′,X′0C′, or X′0E′) and the first subfield must be Software Program Product Number (X′08′) containing the 7 Character Program Product Number (the PID Number).
  2. All product classifications are supported (X′01′, X′03′, X′04′, X′09′, X′0C′, and X′0E′).
  3. Only software subfields X′04′, X′06′, and X′08′ are supported.
  4. Only hardware subfields X′00′ (format type X′11′ required), and X′0E′ are supported.

- X′91′ - Basic Alert - only one allowed per NMVT. See "Tables of Text for X′91′ Subvector Support" for more information Alert type, general cause, and specific component codes.

Note: Place the Alert description code value in bytes 7-8 into bytes 9-10 and 11-12.

Optional subvectors are:

- X′00′ - Text Message - only one allowed per NMVT. All text is translated from ASCII to EBCDIC by NetView/PC before it is sent to the host. Maximum length allowed is 160 bytes.
- X′A0′ - Detail Qualifier - three maximum per NMVT. Maximum length allowed is 8 bytes. Detail Qualifiers are used only by the host. They appear on the Event Detail panel.

Required subvector for local Alert Manager NMVTS:

- X′9F′ - NetView/PC Alert subvector

Any other Subvectors included in the Major Vector will **not** be processed by NetView/PC and will **not** be sent to the host.

An example of a NetView/PC Alert Major Vector showing the required and optional subvectors is:

Length	X'0000'	X'01'	X'03'	X'10'	X'91'	X'10'	X'00'	X'9F'	X'A0'

Figure 35. NetView/PC Non-generic Alert NMVT Example

Tables of Text for X′91′ Subvector Support

Alert Type

The Alert Type indicates the severity of the Alert. Examples are PERMANENT, TEMPORARY. The Alert type is displayed on the Static Alerts and Selected Alert Details panels.

This file is indexed by byte 3 of the Basic Alert Subvector.

The records of this key-sequenced data set are 11 bytes long. The first byte is the key (the index) and the remaining 10 bytes are text description. The text description is compressed to 10 characters for inclusion on the Static Alerts and Dynamic Alerts panels.

For readability, blanks are shown between the key and the text. Blanks are not included in the file.

Key	Text	Description
01	PERM ERROR	A loss of availability to the end user that is not recovered from without intervention external to the reporting product.
. . .		
0F	DELAYED	The sender is reporting a previously detected Alert condition that prevented reporting when detected.
		See *System Network Architecture Formats* for a list of Alert types.

Figure 36. SV X'91' Alert Type Field, File : DUPALATF.TXT

General Cause Table

The general cause is included on the Dynamic Alerts panel. It is the 4th byte of the Basic Alert subvector. This text is displayed on the Static Alerts and Selected Alert Details panels.

Key	Text
01	HARDWARE OR MICROCODE (NOT DISTINGUISHED)
. . .	
18	MICROCODE OR SOFTWARE (NOT DISTINGUISHED)
	See *System Network Architecture Formats* for a complete list of cause codes.

Figure 37. SV X'91' Cause Code Field, File : DUPALGCF.TXT

Specific Component Table

The text for the specific component is displayed on the Dynamic Alerts and Static Alerts panels. Bytes 5-6 of the Basic Alert subvector index this Table. Examples from this table are DASD DEVICE, LINK:COMMON CARRIER, LOCAL MODEM, REMOTE MODEM.

This file is indexed by bytes 5-6 of the Basic Alert Subvector.

The records of this key-sequenced data set are 64 bytes long. The first 2 bytes are the key (the index) and the remaining 62 are text. This text is displayed on the Static Alerts and the Dynamic Alerts panels. It may be truncated on the Dynamic Alerts, because only 43 characters are allowed for the display of Device data on that panel, but it will be displayed in full on the Static Alert Panel.

For readability, a blank is shown between the key and the text. This blank is not included in the file. The truncation point for Dynamic Display panel is shown in this list.

1 2 Key	3456789012345678901234567890123456789012345 6789012345678901234 Text
0001	BASE PROCESSOR
. . .	
001F	X.21 LINK CONNECTION EXTERNAL TO THIS PRODUCT
	See *System Network Architecture Formats* for a list of the documented component codes. The following additional component codes are supported.
0080	TOKEN-RING LAN ERROR
0081	CARRIER SENSE MULTIPLE ACCESS (CSMA/CD) LAN ERROR
00F0	COMPUTERIZED BRANCH EXCHANGE (CBX)
00F1	PROCESSOR
00F2	TRUNK
00F3	TERMINAL EQUIPMENT
00F4	ROLM APPLICATION
00F5	T1 RESOURCE MANAGER
00F6	PRIVATE BRANCH EXCHANGE (PBX)

Figure 38. SV X'91' Specific Component Code Field, File : DUPALSCF.TXT

NetView/PC X'9F' Subvector

The NetView/PC Alert X'9F' subvector must be built as described in this section. Use ASCII unless hex (X'nn') is specified and use the Intel (PC) format for fixed(16) fields.

length	X'9F'	Flags	Reserved	Alert Desc.	Prob. Cause sf

Prob. Cause sf	User Cause sf	Install Cause sf	Failure Cause sf

The 'Cause' subfields contain:

1. One or more cause code points (indices into tables of text).
2. One or more recommended action code points.
   See Figure 51 on page 86.

Figure 39. NetView/PC Alert. Format of NetView/PC Alert

Field	Description
Length	One (1) byte - NetView/PC Alert Subvector length.
X'9F'	One (1) byte - NetView/PC Alert Subvector key.
Reserved	Three (3) bytes - Must be zero.
Desc- ription	Two (2) bytes - NetView/PC Alert Description Code Point. NetView/PC uses the Alert Description Code point to get the ALERT DESCRIPTION it displays on the Dynamic Alerts panel, the Static Alerts panel, the Selected Alert Details panel and the Alert Recommended Actions panels. See Figure 46 on page 84.

Figure 40 (Part 1 of 2). NetView/PC X'9F' Subvector fields.

Field	Description
Subfield	One or more subfields as described in Figure 41.

Figure 40 (Part 2 of 2). NetView/PC X'9F' Subvector fields.

The following subfields in the X'9F' subvector may be in any order. Each subfield is required once. If more are present they are ignored. Multiple code points (10 max) may be placed in a subfield.

Subfield	Description
Length	One (1) byte - Probable Cause subfield length.
X'01'	One (1) byte - Probable Cause Subfield Key;
code point	One or more 2 byte Probable Cause code points. The NetView/PC Probable Cause(s) are displayed on the Selected Alert Details Panel. When more than one probable cause code point is included in the Alert, it is the responsibility of the sending product to ensure that they are listed in the sequence of probability. See Figure 47 on page 84 for code point values.
Length	One (1) byte - User Cause subfield length.
X'02'	One (1) byte - User Cause Subfield Key.
count	One (1) byte count of User Cause code points.
code point	One or more 2 byte User Cause code points. The NetView/PC User Cause text advises the operator of conditions which may have caused the Alert which he can resolve without contacting any service organization. If there are no user causes, the cause code of NONE should be included in the Alert. If there are more than one, it is the responsibility of the sending product to include the codes in order of probability. This data is displayed on the Alert Recommended Actions Panel. See Figure 48 on page 85 for code point values.
Length	One (1) byte - Install Cause subfield length.
X'03'	One (1) byte - Install Cause Subfield Key.
count	One (1) byte count of Install Cause code points.
code point	One or more 2 byte Install Cause code points. The NetView/PC Install Cause text identifies installation errors and provides the NetView/PC terms to be used for each condition. If no installation caused conditions apply to this Alert, the code point for NONE should be included. If multiple installation causes are included, the sending product must include them in the sequence of probability. This data is displayed on the Alert Recommended Actions Panel. See Figure 49 on page 85 for code point values.
Length	One (1) byte - Failure Cause subfield length.
X'04'	One (1) byte - Failure Cause Subfield Key.
count	One (1) byte count of Failure Cause code points.
code point	One or more 2 byte Failure Cause code points. The NetView/PC Failure Cause text defines failing components. If more than one code point is provided indicating that more than one component could have caused the Alert condition, the sending product must insure that the code points are sequenced in the Alert so that the highest probability component is first and the last code point is the least probable. This data is displayed on the Alert Recommended Actions Panel. See Figure 50 on page 86 for code point values. **Note:** For User Cause, Install Cause, and Failure Cause, NetView/PC Recommended Action terms are used to provide the operator with an appropriate list of recommended actions that should be followed to resolve this alerted condition. This data is displayed on the Alert Recommended Actions Panel.

Figure 41. NetView/PC X'9F' Subfields.

NetView/PC Cause Subfields

The following figures describe the details of the Probable Cause, User Cause, Install Cause, and Failure Cause subfields of the X′9F′ subvector.

0	1	2&3		⟶ p
p+1	X'01'	Probable Cause code point	...	Probable Cause Code point

The fields of this subfield are:

Byte	Description
0	Length of entire subfield - p+1. Minimum length is 4. The Number of Probable Cause Code points can be determined by subtracting 2 and dividing by 2.
1	Key X'01'
2-p	One or more code points to index probable cause table. Each code point is 2 bytes long. The first byte indexes the default NetView/PC probable cause. The second byte indexes the replacement NetView/PC probable cause. Note that NetView/PC can handle a maximum of 10 code points. If more are present they will be ignored.

Figure 42. Probable Cause subfield of NetView/PC Alert Data Subvector

0	1	2	3 & 4 ⟶ (2n+2)	(2n)+3 & 4		⟶ q
q+1	X'02'	#of user causes	u.c.#1 ... u.c.#n	rec.act#1	...	rec.act#x

The format of this subfield is as follows:

Byte	Description
0	Length of entire subfield - q+1. Minimum length is 5 bytes.
1	Key - X'02'
2	Number (n) of user causes in this subfield. The minimum for n is 1. Note that NetView/PC can handle a maximum of 10 code points. If more are present they will be ignored.
3 thru (2n+2)	User Cause Code points (2 bytes each)
(3+2n) thru q	Recommended Action Code Points (2 bytes each) The number of Recommended Action Code Points can be by determined by subtracting 2*n from q†2 and dividing by 2. (NetView/PC will only handle a maximum of 10 code points.)

Figure 43. User Cause Subfield of the NetView/PC Alert Data Subvector

0	1	2	3 & 4 → (2n+2)			(2n)+3 & 4 → r		
r+1	X'03'	#of inst.causes	i.c.#1	...	i.c.#n	rec.act#1	...	rec.act#x

The following is a description of the bytes of this subfield:

Byte	Description
0	Length of entire subfield - r+1. Minimum length is 5 bytes.
1	Key - X'03'
2	Number (n) of install causes in this subfield. If there are no install causes the code point for "NONE" must be included. (NetView/PC will only handle a maximum of 10 code points.)
3 thru (2n+2)	Install Cause Code points (2 bytes each)
(3+2n) through r	Recommended Action Code Points (two bytes each) The number of Recommended Action Code Points can be determined by subtracting 2*n from r-2 and dividing by 2. (NetView/PC will only handle a maximum of 10 code points.)

Figure 44. Install Cause Subfield of the NetView/PC Alert Data Subvector

0	1	2	3 & 4 → (2n+2)			(2n)+3 & 4 → s		
s+1	X'04'	#of fail.causes	f.c.#1	...	f.c.#n	rec.act#1	...	rec.act#x

The following is a description of the bytes in this subfield:

Byte	Description
0	Length of entire subfield - s+1. Minimum length is 5 bytes.
1	Key - X'04'
2	Number (n) of failure causes in this subfield. If there are no failure causes the code point for "NONE" must be included. (NetView/PC will only handle a maximum of 10 code points.)
3 thru 3+(2n-1)	Failure Cause Code points (2 bytes each)
(3+2n) thru s	Recommended Action Code Points (2 bytes each) The number of Recommended Action Code Points can be determined by subtracting 2*n from s-2 and dividing by 2. (NetView/PC will only handle a maximum of 10 code points.)

Figure 45. Failure Cause Subfield

NetView/PC ALERT SV X'9F' Code Point File : DUPALGTF.TXT

The following tables have blanks between the fields of the records. The blanks are added for readability and are not in the file.

The numbers across the page above each list of records indicates byte positions. The first 5 bytes are hex representations and 6 to the end are ASCII characters.

ALERT Description Records

Type 1	Code Point 2-3	Seq. no. 4	Cont-inue 5	6 through 67 (62 character message)
01	0100	01	FF	ABNORMAL TERMINATION
01	0200	01	FF	ACCESS ERROR
01	0300	01	FF	ACTIVATION ERROR
01	0400	01	FF	ADDRESS ERROR
01	0500	01	FF	BEACON ERROR
01	0600	01	FF	BUFFER ERROR
01	0700	01	FF	BUS ERROR
01	0800	01	FF	COMMAND REJECTED
01	0900	01	FF	CONNECTION ERROR
01	0A00	01	FF	DATA READ ERROR
01	0B00	01	FF	DATA WRITE ERROR
01	0C00	01	FF	EQUIPMENT MALFUNCTION
01	0D00	01	FF	INTERVENTION REQUIRED
01	0E00	01	FF	LOST DATA ERROR
01	0F00	01	FF	NOTIFICATION
01	1000	01	FF	OVERRUN ERROR
01	1100	01	FF	PERFORMANCE DEGRADED
01	1200	01	FF	POWER LOSS
01	1300	01	FF	PROCEDURAL ERROR
01	1400	01	FF	PROGRAM ABEND
01	1500	01	FF	PROGRAM ERROR
01	1600	01	FF	PROTOCOL ERROR
01	1700	01	FF	SPECIFICATION ERROR
01	1800	01	FF	THERMAL ERROR
01	1900	01	FF	THRESHOLD EXCEEDED
01	1A00	01	FF	TIMEOUT ERROR
01	1B00	01	FF	UNDERRUN ERROR
01	1C00	01	FF	UNDETERMINED ERROR

Figure 46. X'9F' subvector Alert Description records

Type 1	Code Point 2-3	Seq. no. 4	Cont-inue 5	6 through 67 (62 character message)
02	0100	01	FF	ADAPTER
02	0200	01	FF	APPLICATION PROGRAM
02	0300	01	FF	ATTACHMENT
02	0400	01	FF	CABLE
02	0500	01	FF	CALLED NUMBER
02	0600	01	FF	CAPACITY EXCEEDED
02	0700	01	FF	CHANNEL

Figure 47 (Part 1 of 2). Probable Cause records

Type **1**	**Code Point** **2-3**	**Seq. no.** **4**	**Cont-inue** **5**	**6 through 67 (62 character message)**
02	0800	01	FF	COMMUNICATIONS
02	0900	01	FF	CONGESTION
02	0A00	01	FF	CONTROL PROGRAM
02	0B00	01	FF	CONTROLLER
02	0C00	01	FF	DATA
02	0D00	01	FF	DEFINITION
02	0E00	01	FF	DEVICE
02	0F00	01	FF	FUNCTION NOT SUPPORTED
02	1000	01	FF	LINE
02	1100	01	FF	MEDIA
02	1200	01	FF	MODEM
02	1300	01	FF	NONE
02	1400	01	FF	PROCESSOR
02	1500	01	FF	STORAGE
02	1600	01	FF	UNAUTHORIZED
02	1700	01	FF	UNDETERMINED
02	1800	01	FF	USER

Figure 47 (Part 2 of 2). Probable Cause records

Type **1**	**Code Point** **2-3**	**Seq. no.** **4**	**Cont-inue** **5**	**6 through 67 (62 character message)**
03	0100	01	FF	CABLE NOT CONNECTED
03	0200	01	FF	CONNECTION NOT ESTABLISHED
03	0300	01	FF	CONTAMINATION
03	0400	01	FF	DUMP REQUESTED
03	0500	01	FF	FILE FULL
03	0600	01	FF	INCORRECT PROCEDURE
03	0700	01	FF	INTERVENTION REQUIRED
03	0800	01	FF	LINE NOT ENABLED
03	0900	01	FF	MEDIA
03	0A00	01	FF	MEDIA JAM
03	0B00	01	FF	MEDIA SUPPLY EXHAUSTED
03	0C00	01	FF	NONE
03	0D00	01	FF	NORMAL CONDITION
03	0E00	01	FF	OPERATOR GENERATED
03	0F00	01	FF	OFF LINE
03	1000	01	FF	POWER OFF

Figure 48. User Cause records

Type 1	Code Point 2-3	Seq. no. 4	Cont-inue 5	6 through 67 (62 character message)
04	0100	01	FF	CABLE CONNECTION INCORRECT
04	0200	01	FF	FUNCTION NOT PERMITTED
04	0300	01	FF	INCORRECT HARDWARE CONFIGURATION
04	0400	01	FF	INCORRECT SOFTWARE GENERATION
04	0500	01	FF	MISMATCH BETWEEN HARDWARE AND SOFTWARE
04	0600	01	FF	NONE

Figure 49. Install Cause records

Type 1	Code Point 2-3	Seq. no. 4	Cont-inue 5	6 through 67 (62 character message)
05	0100	01	FF	ADAPTER
05	0200	01	FF	APPLICATION PROGRAM
05	0300	01	FF	CABLE
05	0400	01	FF	CHANNEL
05	0500	01	FF	COMMUNICATIONS
05	0600	01	FF	CONTROL PROGRAM
05	0700	01	FF	CONTROLLER
05	0800	01	FF	DATA
05	0900	01	FF	DEVICE
05	0A00	01	FF	LINE
05	0B00	01	FF	MEDIA
05	0C00	01	FF	MODEM
05	0D00	01	FF	NONE
05	0E00	01	FF	PROCESSOR
05	0F00	01	FF	STORAGE

Figure 50. Failure Cause records

Type 1	Code Point 2-3	Seq. no. 4	Cont-inue 5	6 through 67 (62 character message)
06	0100	01	FF	CONTACT APPROPRIATE SERVICE REPRESENTATIVE
06	0200	01	FF	CORRECT INSTALLATION PROBLEM
06	0300	01	FF	CORRECT AND RETRY
06	0400	01	FF	IF PROBLEM PERSISTS THEN DO THE FOLLOWING
06	0500	01	FF	IF PROBLEM RECURS THEN DO THE FOLLOWING
06	0600	01	FF	PERFORM PROBLEM DETERMINATION PROCEDURES
06	0700	01	FF	PERFORM PROBLEM RECOVERY PROCEDURES
06	0800	01	FF	OBTAIN DUMP
06	0900	01	FF	REVIEW DETAIL DATA
06	0A00	01	FF	RUN APPROPRIATE TEST
06	0B00	01	FF	RUN APPROPRIATE TRACE

Figure 51. Recommended Action records

Generic Alert Format

Required subvectors in generic alerts[5] are:

- X'01' - Date/Time - only one allowed per NMVT. Only local Date/Time subfield X'10' is supported.
- X'05' - Hierarchy/Resource List - only one allowed per NMVT. Only the Hierarchy Name List X'10' subfield is supported.
    1. The Hierarchy complete indicator bit (bit zero) can be set as described below:
       When zero, NetView will only use the resource names and types in this (HNL) subfield.
       When one, NetView will concatenate the first two resource names and types in this (HNL) subvector with three VTAM resource names and types (PU name, link, and controller).
    2. Five Maximum resource names and types are allowed in the HNL subfield.
    3. First Hierarchy Names List Entry must contain the resource (NetView/PC or PU) name with the type identifier : 'SP ' (Service Point). The resource name is located in DOS file 'DCJSFSPN.REC'. The file has a single 11 byte record in the following format.
       resource name - Eight (8) bytes
       line control - Three (3) bytes - 0D0A1AH
- X'10' - PSID - two maximum per NMVT.
- X'11' - Multiples allowed in each PSID subvector.
    1. All product classifications are supported (X'01', X'03', X'04', X'09', X'0C', and X'0E').
    2. All software subfields are supported (X'02',X'04',X'06', X'07',X'08',X'09').
    3. All hardware subfields are supported (X'00',X'01',X'0B', X'0E').
- X'92' - Generic alert data — only one allowed per NMVT.
- X'93' - Probable Causes — only one allowed per NMVT.
- Item One (1) or two (2) below:
    1. One or more of the following may be present in any combination.
        - X'94' — User Causes
        - X'95' — Install Causes
        - X'96' — Failure Causes
    2. When this subvector is present, X'94', X'95', and X'96' may not be present.
        - X'97' Cause Undetermined

[5] See *System Network Architecture Formats*.

Optional subvectors are:

- X'31' - Self-Defining Text Message - only one allowed per NMVT. Subfields X'01', X'11', X'21', and X'30' are supported. All text is translated from ASCII to EBCDIC by NetView/PC before it is sent to the host.
- X'51' - LAN Link Connection Subsystem Data - only one allowed per NMVT. Subfields X'02' through X'0A' and X'23', X'24', X'26', and X'28' are supported. All text is translated from ASCII to EBCDIC by NetView/PC before it is sent to the host.
- X'8C' - SDLC Link Station Data - only allowed one per NMVT. Subfields X'01' through X'08' are supported.
- X'98' - Detailed Data - only one allowed per NMVT. All text is translated from ASCII to EBCDIC by NetView/PC before it is sent to the host.
- Network Alert common subfields.

  The following Alert X'0000' common subfields can be used in combination with supported subvectors as documented in the SNA Architecture.

  - X'81' — Recommended Actions
  - X'82' — Detailed Data
  - X'83' — Product Set ID Index

Any other Subvectors included in the Major Vector will **not** be processed by NetView/PC and will **not** be sent to the host.

An example of a NetView/PC generic alert major vector showing the required and optional subvectors is:

Length	X'0000'	X'01'	X'05'	X'10'	X'92'	X'93'	X'94'	X'95'	X'31'

Figure 52. NetView/PC Generic Alert NMVT Example

Appendix C. Service Point Command Data

API/CS Supported NetView Commands

LINKDATA

The LINKDATA command obtains data from a service point.

The format of the LINKDATA command is:

```
LINKDATA      SP=service point name,
              APPL=application name,
              LINE=line name|RESOURCE=resource name
              [,UN=using node|,ENTRYLCC=entry LCC]
              [,RD=remote device (node)|,EXITLCC=exit LCC]
```

where:

SP	specifies the name of the Service Point to execute the command.
APPL	specifies the name of the LCSM to execute the command.
LINE	identifies the linename of the link connection.
RESOURCE	identifies the name of link connection component within a link connection.
UN	identifies the name of the primary link station for an unbalanced mode link or either node that contains the link station of a balanced mode link.
ENTRYLCC	identifies the name of the first (entry) link connection component of a link connection.
RD	identifies the name of the secondary (adjacent) link station for an unbalanced mode link or the other node containing a link station of a balanced mode link.
EXITLCC	identifies the name of the last (exit) link connection component of a link connection.

Usage Notes

ENTRYLCC and EXITLCC can be used to narrow down the data received. This command can be issued from a CLIST to help automate problem determination and error recovery. If LINKDATA is issued from a CLIST, the resulting data is returned to the CLIST for its use. If LINKDATA is issued from a command line, the results are displayed on your terminal on one or more LINKDATA REPLY panels.

Example

To send a LINKDATA command to service point NMWS1 to retrieve data on line LIN3, enter:

```
LINKDATA SP=NMWS1,APPL=APPL07,LINE=LIN3
```

APPL07 is the LCSM that will execute the command.

LINKPD

The LINKPD command requests a service point to do problem determination analysis on a given link or link segment.

The format of the LINKPD command is:

```
LINKPD        SP=service point name,
              APPL=application name,
              LINE=line name|RESOURCEresource name
              [,UN=using node|,ENTRYLCC=entry LCC]
              [,RD=remote device (node)|,EXITLCC=exit LCC]
```

where:

SP	specifies the name of the Service Point to execute the command.
APPL	specifies the name of the LCSM to execute the command.
LINE	identifies the linename of the link connection.
RESOURCE	identifies the name of link connection component within a link connection.
UN	identifies the name of the primary link station for an unbalanced mode link or either node that contains the link station of a balanced mode link.
ENTRYLCC	identifies the name of the first (entry) link connection component of a link connection.
RD	identifies the name of the secondary (adjacent) link station for an unbalanced mode link or the other node containing a link station of a balanced mode link.
EXITLCC	identifies the name of the last (exit) link connection component of a link connection.

Usage Note

ENTRYLCC and EXITLCC can be used to narrow down the data received. This command can be issued from a CLIST to help automate problem determination and error recovery. If LINKPD is issued from a CLIST, the resulting data is returned to the CLIST and to your terminal as a message or messages.

Example

To send a LINKPD command to service point (SP) NMWS1 to do a problem analysis on line LIN3, enter:

```
LINKPD SP=NMWS1,APPL=APPL07,LINE=LIN3
```

APPL07 is the link connection subsystem manager that will execute the command.

LINKTEST

The LINKTEST command requests a service point to test a given link or link segment.

The format of the LINKTEST command is:

```
LINKTEST      SP=service point name,
              APPL=application name,
              LINE=line name|RESOURCE=resource name
              [,UN=using node|,ENTRYLCC=entry LCC.]
              [,RD=remote device (node)|,EXITLCC=exit LCC]
              [,SELFCNT={number of repetitions|1}]
```

where:

SP	specifies the name of the Service Point to execute the command.
APPL	specifies the name of the LCSM to execute the command.
LINE	identifies the linename of the link connection.
RESOURCE	identifies the name of link connection component within a link connection.
UN	identifies the name of the primary link station for an unbalanced mode link or either node that contains the link station of a balanced mode link.
ENTRYLCC	identifies the name of the first (entry) link connection component of a link connection.
RD	identifies the name of the secondary (adjacent) link station for an unbalanced mode link or the other node containing a link station of a balanced mode link.
EXITLCC	identifies the name of the last (exit) link connection component of a link connection.
SELFCNT	specifies the number of self test repetitions to be executed. The range is 1-255, with default = 1.

Usage Note

ENTRYLCC and EXITLCC can be used to narrow down the data received. This command can be issued from a CLIST to help automate problem determination and error recovery. If LINKTEST is issued from a CLIST, the resulting data is returned to the CLIST for its use. If LINKTEST is used from a command line, the results are displayed at your terminal on one or more LINKTEST REPLY panels.

Example

To send a LINKTEST command to service point NMWS1 to perform a test on line LIN3, enter:

```
LINKTEST SP=NMWS1,APPL=APPL07,LINE=LIN3
```

APPL07 is the application that will execute the command.

RUNCMD

The RUNCMD routes commands to service points for execution by one of the service point applications.

The format of the RUNCMD is:

```
RUNCMD      SP=service point name,
            APPL=application name,
            command_string
```

where:

SP	is the network name of the service point which is to receive the given command.
APPL	is the name of the application that is to execute the given command.
command string	is the command to be executed.

Note: The limit on the length of the RUNCMD is 240 characters.

Usage Note

The parameters on the RUNCMD are positional. The given command (command string) must be the last parameter and may be any format.

Example:

```
RUNCMD SP=SP01, APPL=APPL02, DISPLAY LINES
```

Response:

The normal response to RUNCMD will either be message(s) from the service point application or message DSI260I RUNCMD COMPLETE when no messages are returned from the service point application. The messages returned may be command facility or service point application messages.

Service Point Command vectors

This chapter shows the major vectors and the subvectors used for Service Point commands and responses. Subvectors unique to a major vector are shown with the major vector they are used with. Common subvectors are described in "Common Subvectors" on page 102.

You must be familiar with the SNA formats as described in book listed in on page vii, to understand the vectors described in this chapter.

NMVT Length Algorithms

The maximum length of an NMVT supported by NetView/PC is 512 bytes. The following figures describe how to determine the size of an NMVT by figuring the size of overhead and each kind of information contained in the NMVT.

```
Max NMVT length = 512
            - 43
            - 2*(# of Prob Cause variables)

      For each element of this response (LCC NUMBER):
            - 10
            - Resource Type Length (max=8)
            - Resource Name Length (max=8)
```

Figure 53. Max NMVT length possible for the LINKPD "algorithm"

```
Max NMVT length = 512
            - 38

      For each element of this response (LCC NUMBER):
            - 10
            - Resource Type Length (max=8)
            - Resource Name Length (max=8)
         For each LCC data of this element (LCC DATA NUM):
               - 6
               - LCC Data Value Length
               - LCC Data Name Length
```

Figure 54. Max NMVT length possible for the LINKDATA "algorithm"

```
Max NMVT length = 512
        - 54

        For each element of this response (LCC NUMBER):
            - 10
            - Resource Type Length (max=8)
            - Resource Name Length (max=8)
          For each LCC data of this element (LCC DATA NUM):
              - 6
              - LCC Data Value Length
              - LCC Data Name Length
```

Figure 55. Max NMVT length possible for the PUT LINKTEST "algorithm"

NMVT Header

The format of the NMVT header is shown in Figure 56. The header precedes the first major vector of every NMVT.

3	2	2	1
X'41038D'	Ret	PRID	FLAGS

(NMVT Header is described in System Network Architecture Formats

Figure 56. NMVT Header

Service Point Command Major Vectors

The command and response major vectors supported by the API/CS, and their unique subvectors are shown in the following sections. The major vectors and subvectors for each command are shown and are followed by the corresponding response major vectors and unique subvectors.

RUNCMD Vectors

0 1	2 3	4 ——————→ m
m + 1	8061	Subvectors 06 -Name list 31 -Self defining text

Figure 57. RUNCMD

0 1	2 3	4 m
m + 1	0061	Subvectors 44 -Reply count 7D -Sense data

Figure 58. Sense Reply to RUNCMD

0 1	2 3	4 m
m + 1	0061	Subvectors 44 -Reply count
m + 1	1300	Subvectors 0A -Qualified message or 31 -Self defining text

Multiple X'31' and X'0A' subvectors are allowed

Figure 59. Formatted Response message to RUNCMD

0 1	2 3	4 m
m + 1	0061	Subvectors 44 -Reply count
m + 1	1309	Unformatted data

Figure 60. Unformatted Response message to RUNCMD

Unsolicited Operator Message Vectors

0 1	2 3	4 m
m + 1	006F	Subvectors 06 - Name list
m + 1	1300	Subvectors 0A -Qualified message or 31 -Self defining text

Multiple X'31' and X'0A' subvectors are allowed

Figure 61. Send Message To Operator

LINKPD Vectors

0 1	2 3	4 m
m + 1	8062	Subvectors 06 -Name list

Figure 62. LINKPD

0 1	2 3	4 m
m + 1	0062	Subvectors 44 -Reply count 7D -Sense data

Figure 63. Sense Response to LINKPD

0 1	2 3	4 m
m + 1	0062	Subvectors 44 -Reply count
m + 1	130A	Subvectors 82 -Link Segment Status 93 -Probable Cause
m + 1	1307	Subvectors 05 -Hierarchy/Resource List
m + 1	130B	

```
        Code one 1307 MV per resource
Begin Link Connection Component Descriptors Major Vector X'130A'
Link Connection Component Descriptor Major Vector X'1307'
End Link Connection Component Descriptors Major Vector X'130B'
```

Figure 64. Response to LINKPD

0	1	2
p+1	82	Code

```
Status Codes:
  00 -No failure detected
  01 -Detected failure with failing resource isolated
  02 -Detected failure with location not isolated
  03 -Detected failure upstream from managed segment
  04 -Detected failure within the managed segment
  05 -Detected failure downstream from managed segment
```

Figure 65. Link Status Subvector

0	1	2 3 →		p
p+1	93	Code Point	. . .	Code Point

One or more two byte probable cause code point allowed

Figure 66. Probable Cause Subvector

LINKDATA Vectors

0 1	2 3	4 m
m + 1	8063	Subvectors 06 -Name list

Code one or more 80 SV per resource (1307 MV)

Figure 67. LINKDATA

0 1	2 3	4 m
m + 1	0063	Subvectors 44 -Reply count 7D -Sense data

Figure 68. Sense Response to LINKDATA

0 1	2 3	4 m
m + 1	0063	Subvectors 44 -Reply count
m + 1	130A	
m + 1	1307	Subvectors 05 -Hierarchy/Resource List 80 -Link Connection Component Data
m + 1	130B	

```
Begin Link Connection Component Descriptors Major Vector X'130A'
Link Connection Component Descriptor Major Vector X'1307'
End Link Connection Component Descriptors Major Vector X'130B'

         Code one 1307 MV per resource
         Code one or more 80 SV per resource (1307 MV)
```

Figure 69. Response to LINKDATA

0	1	2 3 ⟶ p
p+1	80	Subfields 01 -LCC Name 02 -LCC Hex Value 03 -LCC Character Value 04 -LCC Decimal Value 05 -LCC Bit string

NOTE: Only one of subfields
2 through 5 is allowed

SPCI Parameter LCC Data Subfields X'01' - X'05'

0	1	2 ⟶ q
q+1	01 to 05	Data to be displayed (Decimal data is from 1 to 4 bytes)

Figure 70. LCC data subvector

LINKTEST Vectors

0 1	2 3	4 m
m + 1	8064	Subvectors 06 -Name list 80 -Test set up data

Figure 71. LINKTEST

0	1	2 ⟶ p
p+1	80	Subfields 01 - Test Count

SPCI Self Test Count Subfield X'01'

0	1	2 3
04	01	Test Count

Figure 72. Test Set Up Data Subvector

0 1	2 3	4 m
m + 1	0064	Subvectors 44 -Reply count 7D -Sense data

Figure 73. Sense Response to LINKTEST

Begin Link Connection Component Descriptors Major Vector X'130A'
Link Connection Component Descriptor Major Vector X'1307'
End Link Connection Component Descriptors Major Vector X'130B'

0 1	2 3	4 m
m + 1	0064	Subvectors 44 -Reply count 81 -Link Test Results
m + 1	130A	
m + 1	1307	Subvectors 05 -Hierarchy/Resource List 80 -Link Connection Component Data
m + 1	130B	

Code one 1307 MV per resource
Code one or more 80 SV per resource (1307 MV)

Figure 74. Response to LINKTEST

0	1	2 3 → p
p+1	81	Subfields 01 -Execution Indicator 02 -Test Type 03 -Count Requested 04 -Count Executed

SPCI Parameter Link Test Execution Indicator Subfield X'01'

0	1	2
03	01	Code

00 - Passed
01 - Failed
02 - Indeterminate

SPCI Parameter Link Test Test Type Subfield X'02'

0	1	2
03	02	Code

00 - Background self test
01 - Immediate self test

SPCI Parameter Link Test Count Requested Subfield X'03'

0	1	2 3
04	03	Count

SPCI Parameter Link Test Count Executed Subfield X'04'

0	1	2 3
04	04	Count

Figure 75. Link Test Results Subvector

0	1	2 3 ⟶ p
p+1	80	Subfields 01 -LCC Name 02 -LCC Hex Value 03 -LCC Character Value 04 -LCC Decimal Value 05 -LCC Bit string

NOTE: Only one of subfields
2 through 5 is allowed

SPCI Parameter LCC Data Subfields X'01' - X'05'

0	1	2 ⟶ q
q+1	01 to 05	Data to be displayed (Decimal data is from 1 to 4 bytes)

Figure 76. LCC data subvector

Common Subvectors

The subvectors in this section have the same (common) meaning and use wherever they appear in NMVTs.

0	1	2 3 ⟶ p
p+1	05	Subfields 01 -LCC Identification

SPCI Parameter LCC Identification Subfield X'01'

0	1	2 ⟶ q
q+1	01	LCC Type ⅞ LCC Name

SPCI Parameter LCC Type or LCC Name Entry

0	1 ⟶ r
r+1	Name (<= 8 chars)

LCC Type should be standard nomenclature

Figure 77. Hierarchy/Resource List Subvector

0	1	2 3 ⟶ p
p+1	06	Subfields 01 - Link Segment List 50 - Application Name

SPCI Link Segment List Subfield X'01'

0	1	2 3 q
q+1	01	List Entry

Note: Not used with 006F or 8061 Major Vectors

SPCI Link Segment List Entry

0	1 2 3 4 5 6 78
r	LCC name (length of r (r= 1 to <= 8)

Multiple List Entries Allowed

SPCI Application Name Subfield X'50'

0	1	2 ⟶ q
q+1	50	Name (<=8 chars)

Figure 78. Name List Subvector

0	1	2	3 → p
p+1	0A	Subfields 01 -Message ID 02 -Replacement Data	

SPCI Parameter Message ID Subfield X'01'

0	1	2	3	4	5	6	7	8
09	01	Formatted Message ID						

format: aaannnn where a=alpha n=numeric
it will be used to access the host message table

SPCI Parameter Replacement Text Subfield X'02'

0	1	2 → q
q+1	02	Text

02 subfields must equal the expected number defined
in the host message

Figure 79. Qualified Message Subvector

0	1	2 → p
p+1	31	Message text (240 bytes max)

Figure 80. Text Message Subvector

0	1	2 3 ——→ p
p+1	44	Subfields 01 -Reply Count 10 -Buffer count 11 -Max RU size

Reply Count Subfield X'01'

0	1	2 3
04	01	00 01

Buffer Count Subfield X'10'

0	1	2 3 4 5
06	10	00 00 00 01

Max RU size Subfield X'11'

0	1	2 3 4 5
06	10	00 00 02 00

Figure 81. Reply Count Subvector

0	1	2 3 4 5
06	7D	Code 1 ⅞ Code 2

Figure 82. Sense Data Subvector

Appendix D. Suggested Command Formats

Suggested Physical Device Management Commands

The following descriptions provide a suggested set of commands for common functions needed in telecommunications device management. They fall into verbs for the management of physical devices and verbs for the management of the data bases used to track the configuration of the physical devices. There is also a verb for encapsulating those commands not covered in the other two categories.

Following the verb descriptions is a section suggesting the encoding of the verbs in a language free manner.

Note: NetView CLISTs will only accept eight character names.

LINK—CHANGE

LINK–CHANGE changes the connection between two resources where such a connection can be changed. It is used in environments such as a matrix switch to establish or disconnect a connection or in a multiplexer to connect a port to a slice of the available band width or reset that connection. LINKCHNG is the primary command for effecting the physical network changes which are needed in recovery actions. LINKCHNG SP=service point name
FROM=resource 1 name
TO=resource 2 name
ACTION={CONNECT|DISCONN}

Purpose of Command

This command is used to change the connectivity relationships between existing physical resources.

Actions Taken by Receiver

The named connection is to be made or broken according to the action requested.

Inputs:

ACTION	CONNECT establishes a connection between the FROM and TO resources. DISCONN disconnects the FROM and TO resources. This parameter is required.
FROM	resource 1 name - the name of the first resource in the pair whose connectivity is to be changed.
TO	resource 2 name - the name of the second resource in the pair whose connectivity is to be changed.

Outputs:

Return code	Indication of whether the command was successful or not.

LINK—DISPLAY

LINK-DISPLAY would cause the transmission of stored or actively collected data to be forwarded as a reply. The application requesting this data would have to be intimately familiar with the device for which the data is reported, but such an application would consequently be able to make the most detailed decisions regarding the actions to take for that device.

The data collected could be error data, response time data, accounting data, etc.
LINKDISP SP=service name point
LINE=(line name) nknm1

Purpose of Command

This command provides a means of collecting device control, statistics, and error data for a particular resource. This capability allows detailed problem determination of a resource to be performed by an operator or CLIST. The CLIST usefulness is reduced, due to the nature and amount of data being returned, so that only the success or failure of the command is available to the CLIST.

Actions Taken by Receiver

Upon receipt of this command, the receiver will gather the requested data for the resources within its scope of control, returning the data gathered in a self-defining format.

Inputs:

nknm2

Outputs:

Source information	information indicating the name of the Service Point and the resource to which the data pertains.
self-defining data	data in the form of doublets containing the name of the field being returned and the value of the field being returned. An attribute type and a length will be associated with each of these items to allow the sender to interpret the information received on the reply.

Configuration Data Base Management Commands

RESOURCE—DISPLAY

RESOURCE-DISPLAY requests the return of data from the data base that a product uses to track the physical status of a link component. Like its counterpart LINKDISP, RESDISP returns device dependent data. Unlike LINKDISP, RESDISP returns the state remembered rather than the state which is interrogated.

This is effective in identifying those cases where the program and the device get out of synchronization. It is also effective in recovering information the device does not allow to be queried or which the application uses but does not actually provide to the device interface.
RESDISP SP=service point name
RESOURCE=resource name
NAME=field name

Purpose of Command

This command provides the capability to retrieve information stored by the receiver for a particular resource.

Actions Taken by Receiver

The receiver locates the detail information for the specified resource, formats the information into a self-defining format, and replies to the sender with the self-defining data.

Inputs:

RESOURCE	the name of the resource for which the field names are to be displayed.
NAME	the name of the field in the database record to be displayed. If no NAME parameter is specified, the entire detail information available for the specified resource will be returned.

Outputs:

Source information	information indicating the name of the Service Point and the resource to which the data pertains.
self-defining data	data in the form of doublets containing the name of the field being returned and the value of the field being returned. An attribute type and a length will be associated with each to allow the sender to interpret the information received on the reply. An indication that the requested field's value was not retrieved will be sent in the value portion of the doublet if the field name is not known by the receiver.

RESOURCE—CHANGE

RESOURCE–CHANGE provides for the case where the data base provided for management of a device needs to be updated. In the previous case where the device and the data base were out of synchronization, this command can restore the data base. (LINKCHNG can be used where the device is to be reset to match the data base.)

The use in recovery is to reconfigure a network to bypass errors.

```
RESCHNG SP=service point name
RESOURCE=(resource name)
NAME=(field name)
VALUE=(field value)
```

Purpose of Command

This command will change parameters regarding a resource. Any parameter that is known by the receiver for the particular resource may be changed, even if the parameter was not previously initialized.

Actions Taken by Receiver

The receiver will locate the detail information using the specified resource name and convert the value specified into a format defined for the field by structures stored in the receiver.

Inputs:

RESOURCE	the name of the resource for which the field names are to be changed.
NAME	the name of the field in the database record to be changed.
VALUE	the value to assign to the field specified by NAME.

Outputs:

Return code	Indication of whether the command was successful or not.

PATH—DISPLAY

PATH–DISPLAY is useful where the application is maintaining the names of multiple link components and the connections among them. The meaning of the PATH is derived from the SNA line model. The one end of the SNA line model is the USING NODE, ordinarily an NCP, and the other end is known as the ADJACENT LINK STATION, usually the cluster controller or terminal end of the line. The PATH concept describes the components which can be identified on the link between the SNA end points. (In some cases multiple paths can be identified between these points.)

PATHDISP is intended to allow a recovery process to identify the applications known to be managing components on a link and to use the link management commands to effect recovery.
PATHDISP SP=service name point nknm1

Purpose of Command

This command displays the path information related to the names provided on the invocation.

Actions Taken by Receiver

The receiver of this command will retrieve configuration path information such as component names, their status and connectivity, their type and machine identification, and their managing applications. This information will be formatted for transmission to the requestor. The information returned should be put into NCCF variables if requested from a CLIST.

Inputs:

nknm2

Outputs:

Resource name	name of a link connection component
Manager name	fully-qualified name of the Service Point application responsible for the resource.
Class	generic type of device: modem, statmux, matrix switch, etc.
Machine type	model number of the device: 3728, 3710, etc.
Status	current status of the device: active, inactive, spare, defective, etc.

PATH—CHANGE

PATH–CHANGE provides for the restructuring of the connections in a data base which is keeping track of the components on a link. It is used in conjunction with the LINKCHNG command. LINKCHNG alters the physical connections and PATHCHNG alters the data base tracking those connections.

The use in recovery is to reconfigure a network to bypass errors.
PATHCHNG SP=service point name
FROM=(resource 1 name)
TO=(resource 2 name)
[ACTION={CONNECT|DISCONN}]
[FROMDISP=disposition]
[TODISP=disposition]

Purpose of Command

This command is used to change the connectivity relationships between existing resource definitions.

Actions Taken by Receiver

The receiver will accept the command and take the action requested by altering the data base connections.

Inputs:

ACTION	CONNECT establishes a connection between the FROM and TO resources. DISCONN disconnects the FROM and TO resources. This parameter is required.
FROM	resource 1 name - the name of the first resource in the pair whose connectivity is to be changed. This parameter is required.
FROMDISP	the status to assign to the resource after its connectivity has been changed. This parameter is optional and defaults to no change in the resource's status.
TO	the name of the second resource in the pair whose connectivity is to be changed. This parameter is required.
TODISP	the status to assign to the resource after its connectivity has been changed. This parameter is optional and defaults to no change in the resource's status.

Outputs:

Return code	Indication of whether the command was successful or not.

Appendix E. Panel Development Rules

Applicability and Conformance

The rules in this chapter were used by the NetView/PC developers to design the NetView/PC panels. They provide instructions for the development of a consistent NetView/PC user interface. The NetView/PC panels were developed with the EZ-VU Screen Definition Facility (SDF), and it is recommeneded for your dialog management panel development. Whether or not you use EZ-VU, following these rules will help achieve a NetView/PC consistent user interface.

Requirements

The panel developer is encouraged to observe the following principles when designing NetView/PC panels:

1. Locate panel elements consistently and in a standard format that is familiar to the user.
2. Reduce the number of user keystrokes and the need for memorization whenever possible.
3. Prompt for an explicit confirmation from the user if information will be lost or destroyed with a requested action (for example, requesting QUIT during update of a record).
4. Insure that all panel input (mixed case or upper case) is "folded" to upper case before acted upon by an application; a simple panel field definition option will perform this function automatically.

Panel Design

On the typical IBM PC display screen of 25 lines by 80 characters, only the first 24 lines can be defined to EZ-VU; the 25th line is managed by NetView/PC as a "workstation status line". Therefore in the following discussion of panel development rules, references to any of the first 24 lines implies the use of EZ-VU for definition; references to line 25 are requirements on NetView/PC for display purposes.

A panel is a particular arrangement of data used to display information to the user, or receive information from the user. The set of generic panel formats presented in this section provides users with a consistent method for making choices or entering data regardless of the particular task being performed.

"Panel Design" includes the following sub-sections:

- "Types of Panel Elements" on page 114 -

  The components (Entry Fields, Selection Fields, Protected Fields and White Space) defined in this sub-section are basic to all panel design.

- "Common Panel Elements" on page 115 -

  The elements (Panel ID, Panel Title, Location Information, Data Set Name Separator, Top and Bottom Environment Areas, Message Line, Command/Selection Line, and Workstation Status Line) defined in this sub-section apply across all panel formats.

- "Panel Body Elements" on page 120 -

  The components (Top and Bottom Instruction Areas, Headings, Key Phrase and Key Phrase ID, Selection Fields, Entry Fields and Explanatory Text) defined in this sub-section apply to the individual panel types.

- "Panel Types" on page 127 -

  Four panel types are defined in this sub-section. Their type is based on the functions to be performed and the body elements they contain.

- "Mixing Panel Types" on page 135 -

  A mixed panel is one that contains the panel body elements of two or more panel types. Rules and guidance for mixing panel bodies are given.

Types of Panel Elements

All panels are constructed from some combination of four basic elements:

1. Protected Fields,
2. Entry Fields,
3. Selection Fields, and
4. White Space.

Protected Fields provide read†only information to the user for status, instructions, definitions, etc. *Entry Fields* and *Selection Fields* are provided for user input and choices in the dialog. *White Space* is "blank" space that does not fall into one of the defined field categories.

Protected Fields: A Protected Field is a field that cannot be changed by the user. For example, the Panel Title element is a protected field.

Entry Fields: An Entry Field is a field into which the user may enter information via a keyboard.

Entry Fields may be fixed or variable in length. The length is defined by the application. Entry Fields are governed by attributes that are further described in "Panel Body Elements" on page 120.

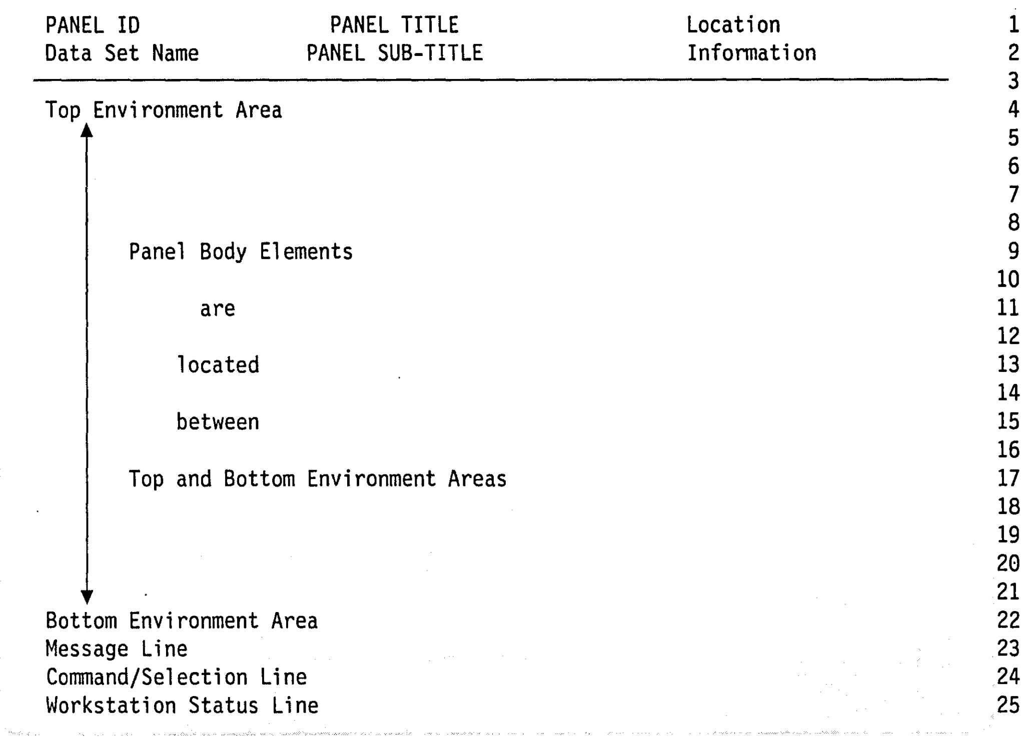
Selection Fields: A Selection Field consists of one or more choices. A choice, itself, is either a Protected Field or an Entry Field.

White Space: White Space makes up the remainder of the panel. It is the panel area that is not occupied by a Protected Field, Entry Field or Selection Field.

White Space is typically used to cause visual separation of information that is presented so that it is readable.

Common Panel Elements

Panels of all types contain common panel elements. The Panel Body contains additional elements that are arranged in various formats (see "Panel Body Elements" on page 120 for details). The following is an example of the placement of the panel elements. The Common Elements include all that are shown in Figure 83. The Panel Body Elements are described within each panel type.



```
  PANEL ID                PANEL TITLE                 Location              1
  Data Set Name         PANEL SUB-TITLE               Information           2
 ______________________________________________________________________     3
  Top Environment Area                                                      4
   ↑                                                                        5
   |                                                                        6
   |                                                                        7
   |                                                                        8
   |      Panel Body Elements                                               9
   |                                                                       10
   |           are                                                         11
   |                                                                       12
   |         located                                                       13
   |                                                                       14
   |         between                                                       15
   |                                                                       16
   |      Top and Bottom Environment Areas                                 17
   |                                                                       18
   |                                                                       19
   |                                                                       20
   ↓                                                                       21
  Bottom Environment Area                                                  22
  Message Line                                                             23
  Command/Selection Line                                                   24
  Workstation Status Line                                                  25
```

Figure 83. Common Panel Elements

The lines around the panel figures in this section represent the boundaries of the panels or the panel bodies and are not part of the panel being described. However, line 3 is part of the panel and represents the "Separator" common element.

As you review each panel type sub-section, remember the placement of these elements.

The Common Panel Elements are described below:

1. Panel Identifier (ID)

   - Purpose - Used for the referencing of a specific panel for diagnostic purposes.

   - Attributes - An alphanumeric, protected field normally eight or fewer characters in length. The Panel ID is located on line 1 of the panel, left justified in upper case.

   - Guidelines for use - A required element. The first three positions of Panel ID must be the unique component prefix assigned to the NetView/PC application; the component prefix is identified in the Product Definition File contained in the application's Distribution Diskette.

2. Panel Title

   - Purpose - It is the name of the panel. The Panel Title may also contain a sub-title to describe the context or current function being performed (for example, EDIT RESOURCE NAME).

   - Attributes - An alphanumeric, protected field centered horizontally on lines 1 and 2 of the panel, in upper case. It must be visibly separated from the other elements on lines 1 and 2.

   - Guidelines for use - A required element. Panel titles and context information should be presented using full words, where possible. Abbreviations may be used only after the abbreviated word was used in an unabbreviated form on a previous panel.

3. Location Information

   - Purpose - If the user is allowed to scroll the data being presented (e.g. resource names in a configuration), the panel must indicate to the user the location or position being viewed relative to the total data available for viewing (for example, "Page 4 of 15").

   - Attributes - An alphanumeric field that contains one or more entry fields or protected fields. It occupies the upper right corner of the panel and is right justified.

   - Guidelines for use - Mandatory when a panel contains data that can be scrolled.

     The phrase "of nnn" is optional; when data from large datasets is displayed, there could be a delay if the number of lines, positions, items, etc., "of nnn" were always calculated and displayed. Panel designers should format location information to be readable and of pleasing appearance. Some common-sense practices are recommended to achieve this, such as suppressing leading zeros on numbers, then aligning corresponding labels, hyphens, instances of "of", and numbers. The following types of Location Information are examples:

     - *More Information*: Used when additional data is available to be displayed. Based on the application, the textual information ("Item", "Row", "Page", etc) may change. Some examples:

       ```
       Item x of n   -or-   Path x of n   -or-   Row x of n
       ```

     - *Panel Number*: Used in a multiple-panel dialog. An example:

       ```
               Panel x of n
       ```

     - *Page Number*: Used when text data is presented as in help panels, document processing, or tutorial presentation. An example:

       ```
               Page x of n
       ```

4. Data Set Name

   - Purpose - Identifies an object in use.

   - Attributes - An alphanumeric, protected field presented in mixed case and left-justified on line 2 of the panel (immediately below the Panel ID).

   - Guidelines for use - An optional element; the data set name must be self-describing information that references the object currently being manipulated within the dialog. An example of self-describing information would be the use of "Configuration xxx" as opposed to just "xxx". User terminology must be presented within this area, not system designations.

5. Separator

   - Purpose - Separates the Panel ID, Panel Title, Location Information, and Data Set Name from the Top Environment Area. This allows for easy identification of these elements by the user.

   - Attributes - A protected field. Separation will be achieved with a solid line on line 3 of the panel. If the panel elements normally appearing on line 2 (Data Set Name, Location Information, and Panel Sub-title) are not present, then the solid separator line can appear on line 2.

   - Guidelines for use - A mandatory element when other Common Elements precede it.

6. Top Environment Area

   - Purpose - Used by a task to display information such as the following items:

     - Tutorial information.

     - Task status information (for example, Time of Day).

     - Information that pertains to the data currently being presented (e.g., "Status of Alert Options by Application").

     - Other types of "continuity data", i.e., information regarding the objects that the user has been acting upon in this application and is carried forward from previous panels.

   - Attributes - An alphanumeric area that consists of protected fields and white space. It begins immediately below the Separator Line, that is, typically on line 4 of the panel.

   - Guidelines for use - An optional area. If it is not used, this area becomes "null" to save space on the panel.

7. Bottom Environment Area

   - Purpose - Used by a task to present information to assist the user in proceeding in the dialog.

   - Attributes - An alphanumeric area that consists of protected fields and white space. Presented in the bottom-most area of the panel, just above the Message Line.

   - Guidelines for use - A mandatory element containing, at a minimum, the currently active function keys (see "Function Key Utilization" on page 144). The following rules for display will provide a consistent method for presenting the active keys for a given panel.

     - display only the active keys for the panel

     - display the "hard" keys to the left of the "soft" keys

     - when "scrolling" keys are active, it is not required to display them

     - display the "soft" keys in numerical order, left to right

     - use the "key=action" format only for "soft" keys

     - when possible, avoid abbreviations of key actions

- for the F4 (Return), use "Main Menu" for the action description except on help panels use F4=Help Main Menu
- use one or more lines to display the keys
- when multiple rows are used, maintain a column format

The following examples will serve to illustrate the above guidelines:

```
Example 1:

  Enter  F1=Help  F3=End


Example 2:

  Enter          F1=Help        F2=Quit    F3=End
  F4=Main Menu   F5=Redisplay   F6=Add     F7=List
```

8. Message Line

   - Purpose - Used for the presentation of "immediate" messages and prompts that are necessary for the user to interact with one task.
   - Attributes - An alphanumeric, protected field displayed in mixed case on line 23.
   - Guidelines for use - A mandatory element, beginning in column 5 of line 23 and extending thru column 80. For details on message structure, see "Messages and Prompts" on page 146.

   Note: When tasks have a need to present a confirmation request prompt to the user, then the prompt is displayed on the message line and the required response is received from the Command/Selection field on line 24. The default response must be displayed in the Command/Selection field so that the user may simply press Enter to accept the default response. For an example, see "QUIT" in "Dialog Control Actions" on page 139.

9. Command/Selection Line

   - Purpose - Used for the entry of commands and selections by the user as well as user responses to message prompts.
   - Attributes - An alphanumeric field displayed in mixed text and left-justified on line 24.
   - Guidelines for use - A mandatory element only when selections, commands or responses to prompts may be entered by the user.

   When this element is presented, it must be identified by the Entry Prompt symbol ===>. Prompt text to the left of the Entry Prompt is required to further identify the type of entry expected. The entry field definition begins in the second position to the right of the arrowhead and may extend to the end of the line. Some examples are:

```
Selection ===> _(Entry prompt with 1-character input
                 field; typically a menu panel)

Command ===> ________(Entry prompt with multi-char input)

'text' ===> _______(Entry prompt with multi-char input;
                    'text', if any, supplied by
                     application)
```

10. Workstation Status Line

    - Purpose - Used to display information pertaining to the operation of the workstation.
    - Attributes - An alphanumeric, protected field displayed in mixed case on line 25 of the screen.
    - Guidelines for use - A mandatory element in the NetView/PC environment; this element is NOT managed by the application but by NetView/PC base services. The Vendor API function Operator Communication provides the interface to this base service.

As a design/development aid, the preceding descriptions of Common Panel Elements have been summarized into a "Reference Chart" as presented in Figure 84. The user of this chart will need to be knowledgeable of the tutorial information before using the chart.

ELEMENT	**ATTRIBUTES**	**LEVEL OF EMPHASIS**
Panel ID	• alphanumeric, protected field • left justified on line 1 • upper case • mandatory element	Level 2
Panel Title	• alphanumeric, protected field • centered horizontally, lines 1 and 2 • upper case • mandatory element	Level 4
Location Information	• alphanumeric field • one or more entry or protected fields • right justified on line 1 or 2 • mixed case • mandatory when panel can be scrolled	Level 3
Data Set Name	• alphanumeric, protected field • left justified on line 2 • mixed case • optional element	Level 3
Separator	• solid line, protected field • line 2 or line 3 • mandatory element	Level 3
Top Environment	• alphanumeric, protected field • begins immediately below Separator • mixed case • optional element	Level 5

Figure 84 (Part 1 of 2). Common Panel Elements

ELEMENT	ATTRIBUTES	LEVEL OF EMPHASIS
Bottom Environment	• alphanumeric, protected field • occupies space immediately above Message Line • contains active F-key assignments • mixed case • mandatory element	Level 5
Message Line	• alphanumeric, protected field • limited to columns 5-80 on line 23 • mixed case • mandatory element	**Information:** Level 7 **Warning:** Level 8 **Error:** Level 9
Command/ Selection Line	• alphanumeric, protected and entry fields • left justified on line 24 • mixed case • mandatory when commands, selections or responses may be entered by user	Level 4
Workstation Status Line	• alphanumeric, protected field • line 25 • mixed case • mandatory element	**Normal:** Level 5 **Information:** Level 7 **Warning:** Level 8 **Error:** Level 9

Figure 84 (Part 2 of 2). Common Panel Elements

Panel Body Elements

Figure 85 shows Panel Body Elements which are located between the Top and Bottom Environment Areas.

```
Top Environment Area
Top Instruction Area

   ID  KEY PHRASE   Entry Field   Explanatory text
   ID  KEY PHRASE   Entry Field   Explanatory text
   ID  KEY PHRASE   Entry Field   Explanatory text
   ID  KEY PHRASE   Entry Field   Explanatory text

Bottom Instruction Area


Bottom Environment Area
```

Figure 85. Panel Body Elements

Each type of panel will utilize these components in a slightly different format. Each panel type is discussed in "Panel Design" on page 113.

The Panel Body Elements are:

1. Top Instruction Area

   - Purpose - Presents instructions to the user on how to make selections or entries within the panel.

   - Attributes - An alphanumeric, protected, field that is located below the Common Elements displaying at the top of the panel. One or more lines presented in mixed-case characters and left-justified to the left margin of the panel; upper case may be used to emphasize a key word or words. Separation of this area from elements above and below is via white space, when space is available.

- Guidelines for use - An optional element, but recommended when multiple interaction techniques are supported within a panel. The instructions must indicate how the user is supposed to interact with the panel. This should be a concise statement, such as: "Select one of these activities:".

2. Headings

   - Purpose - Provide a description of item(s) for readability and clarity of the panel information.

   - Attributes - Protected fields that are visually distinct from the Top Instruction Area and the items to which they refer. White space may be used for this purpose.

     Headings are classified into two categories, *major* and *minor*, and are defined as follows:

     — Minor Headings (column, row or field)

       — Column heading example:

```
HEADING
field 1
field 2
field 3
```

       — Row heading example:

```
HEADING   field 1   field 2   field 3
```

       — Field heading (also called "KEY PHRASE") example:

```
(example of entry (input) field):

                KEY PHRASE. . field


(example of display (output) field):

                KEY PHRASE: field
```

     — Major Heading (also called a "Super Heading"):

```
(example of data/parameter entry (input)):

             MAJOR HEADING
                Minor Heading 1. . field 1
                Minor Heading 2. . field 2
                Minor Heading 3. . field 3


(example of data display (output)):

             MAJOR HEADING
                Minor Heading 1: field 1
                Minor Heading 2: field 2
                Minor Heading 3: field 3


(example of multiple selection fields):

       MAJOR HEADING 1     ID  KEY PHRASE
                           ID  KEY PHRASE
                           ID  KEY PHRASE
                           ID  KEY PHRASE

       MAJOR HEADING 2     ID  KEY PHRASE
                           ID  KEY PHRASE
                           ID  KEY PHRASE
```

All headings must be upper-case except Field Headings (Key Phrases) may be mixed-case on panel types other than Menu and when the number of Field Headings on the panel would cause a readability problem.

- Guidelines for use - An optional element except for "entry fields" and multiple "selection fields", where headings are required. Headings are recommended to enhance clarity and understanding and may be application specific (for example, List Panels and Tabular Data Entry Panels).

3. Key Phrase ID

   - Purpose - Presented to allow the user to select an option by number.
   - Attributes - An alphanumeric, protected, field. When the technique of "selecting items by Key Phrase ID" is used, then the Key Phrase ID must be presented two spaces to the left of the Key Phrase, which is defined below. If a "Single Selection Field Format" on page 127 is used, then Key Phrase IDs may be whole numbers (e.g., 1, 2, 3, etc.) or single alphabetic characters. If a "Multiple Selection Field Format" on page 128 is used, then Key Phrase IDs must be whole numbers prefixed with a single alphabetic character (e.g., A1, A2, A3, B1, C1, etc.). In either case, the IDs must be presented in alphanumeric order followed by any alphabetic IDs in order; the order does not have to remain consecutive. The number zero is not a valid Key Phrase ID. An application should ensure that commonly used choices retain their number; for example, the SHUTDOWN option available from application main menus should always be the ID "S".
   - Guidelines for use - A mandatory element when "selection by Key Phrase ID" is used; indented three spaces from the left margin of the Top Instruction Area.

4. Key Phrase (Minor Field Heading)

   - Purpose - A brief descriptor of a selection choice or a descriptor of an input/output field.

   - Attributes - An alphanumeric, protected, field. This field can be upper case or mixed case, depending on panel usage. For example, on a Menu Panel, the Key Phrases must be upper case; on other panel types, the quantity of Key Phrases should direct the use. For example, when there are many Key Phrases, then mixed case is allowed in order to increase readability; otherwise a single or few Key Phrases must be presented in upper case.

     When preceded by the Key Phrase ID, this field should be presented two spaces to the right of the Key Phrase ID. When the Key Phrase ID is not used, then this field is indented three spaces from the left margin of the Top Instruction Area.

   - Guidelines for use - A mandatory element for Menu Panels and Parameter Entry Panels. The Key Phrase must be meaningful enough so that an experienced user can make a choice without having to refer to any Explanatory Text to the right of the Key Phrase. If the application supports commands, the command name and parameter names should be used as Key Phrases to reinforce learning of the command form of the function.

5. Selection Field

   - Purpose - Used to make a selection from a list of choices.

   - Attributes - Selection Fields contain protected Key Phrase IDs and Key Phrases. Refer to above definitions of these Panel Body Elements. For examples of Selection Fields see Menu Panel under "Panel Types" on page 127.

   - Guidelines for use - The application determines the need for Selection Fields.

6. Entry Field

   - Purpose - An Entry Field is a field within a panel into which the user may enter information via a string input device (for example, a keyboard).

   - Attributes - Entry Fields are "fixed" in length; that is, they have a predetermined length. Input to the field should be left-aligned and the cursor should be positioned at the beginning of the field, ready for entry.

     When panels containing Entry Fields are presented to the user, it is generally helpful to present an indication of how long the field is, and where it is located. NetView/PC panels will be designed to use a Field Length Delimiter for short entry fields and a Field Location Indicator for long entry fields and multi-line entry fields.

     Field Length Delimiter

     A Field Length Delimiter is required for all short entry fields.

     The Field Length Delimiter to be used with NetView/PC panels is the "reverse video" attribute; this attribute can be specified to EZ–VU by assigning the ZATR variable equal to the character string "EW". With the ZATR variable so defined, EZ–VU will apply the reverse video attribute to the field where the cursor is located; when the cursor leaves a field, that field returns to its originally defined attribute and the field receiving the cursor then changes to reverse video.

Field Location Indicator

A Field Location Indicator is required when the Field Length Delimiter is not used; for example, on long entry fields and multi-line entry fields.

The Field Location Indicator to be used with NetView/PC panels is lozenge symbol (■) (ASCII code 254). The lozenge symbol will be placed at the first entry position within an Entry Field.

Information Field versus *Entry Field*

A user should be able to visually distinguish protected fields (information) and modifiable fields (entry). Any protected field that could appear to be an entry field must be presented to the user with a colon (:) delimiter between the description (e.g., the field heading or key phrase) and the information it presents.

Information fields simply present information. For example, the following might appear in an Information Panel:

```
STATUS:  Running
```

This convention may also be utilized with entry fields that were previously completed by the user, and are now presented as information within another panel.

On the other hand, entry fields are input fields; the user is allowed and sometimes required to enter data (default values are almost always presented to the user).

Dot Leadering is to be used to visually connect the Entry Field Heading with the single entry field which it describes. This panel element is primarily found on Parameter and Data Entry panel types (see "Entry Panels" on page 129).

An example of dot leadering for parameter entry:

```
RESOURCE NAME. . ________up to 8 characters
```

An example of dot leadering for data entry:

```
COMPANY
   State. . . __              ZIP. . . . _____
```

In all cases where used, there should be a maximum of **two** dots between the longest heading and the entry field.

When information and entry fields are delimited in the above manner, the user can easily determine what fields are information and what fields may be modified.

- Guidelines for use - Mandatory or optional depending on the panel type; explained as part of each panel type.

7. Explanatory Text

   - Purpose - To explain allowable choices and entries for Selection and Entry Fields.
   - Attributes - An alphanumeric, protected, field arranged in a column exactly three spaces to the right of the longest Selection/Entry field. The text is left justified to that column, and must be self-describing and is presented in mixed case.

- Guidelines for use - An optional element. Should be presented in one of the following ways:
  - A brief description of a menu selection, e.g.,

    Display most recent entries in Problem Directory
  - As a description of the entry field, e.g.,

    Enter up to 8 characters for Resource Name
  - As a range of entry data, e.g.,

    1 to 66 lines
  - As a list of choices, e.g.,

    R=Remote, L=Local
  - As an example of the options, e.g.,

    (show an example character string, as appropriate)

8. Bottom Instruction Area
   - Purpose - Presents instructions to the user concerning what action is necessary after completing the dialog with the panel.
   - Attributes - An alphanumeric, protected, area. Can be floated within the panel, and is placed after the last Selection/Entry Field of the panel, and ahead of the Bottom Environment Area. Presented in mixed case, left-justified and aligned with the Top Instruction Area or left margin of the panel, as appropriate. Multiple lines may be used. Visual separation from other panel elements is achieved with white space.
   - Guidelines for use - An optional element. The instructions must indicate how the user is supposed to continue or end the dialog with the application. This should be a concise statement. This area may also be used to address exceptional or unique action available to the user.

   An example:

   ```
   To update record, press Enter.
   ```

As a design/development aid, the preceding descriptions of Panel Body Elements have been summarized into a "Reference Chart" as presented in Figure 86 on page 126 and Figure 87 on page 126.

The user of this chart will need to be knowledgeable of the tutorial information before using the chart.

ELEMENT	ATTRIBUTES	LEVEL OF EMPHASIS
Top Instruction Area	• alphanumeric, protected field • one or more lines of text left justified to left margin of panel • mixed case • optional element	Level 5
Bottom Instruction Area	• alphanumeric, protected field • one or more lines of text left justified to left margin of panel • mixed case • optional element	Level 5
Key Phrase ID	• alphanumeric, protected field • indented three space (column 4) from left panel margin • always associated with a Key Phrase • a two-char column with alphanumeric characters right-justified in the column, no leading zeros • mandatory only when using the selection technique "Selection by Key Phrase ID"	Level 4
Explanatory Text	• alphanumeric, protected field • left justified and aligned to a margin three spaces to the right of the longest Selection/Entry field • mixed case • optional element	Level 3
Dot Leader to Entry Field	• alphanumeric, protected field • single periods (dots) with no intervening spaces connecting the Key Phrase (Field Heading) to the associated entry field • required element for parameter or data entry fields preceded by a Key Phrase	Level 3

Figure 86. Panel Body Elements

ELEMENT	ATTRIBUTES	LEVEL OF EMPHASIS
Minor Heading - Column and Row	• alphanumeric, protected fields • located above (column) or to the left (row) of two or more input/output fields • upper case • optional element	Level 3[6]
Minor Heading - Field (Key Phrase)	• alphanumeric, protected field • located two spaces to the right of an associated Key Phrase ID, if present; otherwise indented three spaces from: 1. left panel margin, or 2. left margin of its major heading, or 3. right margin of longest entry field located to left of this heading • upper or mixed case, depending on panel usage (other headings on panel) • mandatory for Menu and Parameter Entry panels	Level 3[6]
Major (Super) Heading	• alphanumeric, protected field • location is dependent on panel type and application usage • upper case • optional element	Level 3[6]

Figure 87. Panel Body Element Headings

Panel Types

Menu Panel	Allows the user to choose from a list of related items. ("Menu Panels").
Entry Panel	Allows the user to enter parameters, data, or text. ("Entry Panels" on page 129).
List Panel	Allows the user to manipulate items in a list. ("List Panels" on page 133).
Information Panel	Presents read-only data to the user. ("Information Panels" on page 134).

Menu Panels

Menu Panels provide the user with a set of choices from which the user makes one or more selections. Two Menu Panel types are allowed.

Single Selection Field Format	This format consists of a single selection field. The application can limit the user to a single choice from this panel or may allow the user to make multiple choices from the single selection field.
Multiple Selection Field Format	This format consists of two or more selection fields. The application can limit the user to a single choice from the entire panel, may allow single choices from any or all selection fields, or may allow multiple choices from any or all selection fields.

Single Selection Field Format: Figure 88 presents the Panel Body Elements for Single Selection Field Menu Panels; see Figure 83 on page 115 for location of the Common Panel Elements.

```
Top Instruction Area

   ID  KEY PHRASE   Explanatory Text
   ID  KEY PHRASE   Explanatory Text
   ID  KEY PHRASE   Explanatory Text

Bottom Instruction Area
```

Figure 88. Single Selection Field Menu Panel - Panel Body

The Key Phrases (and Key Phrase IDs) utilized within this panel type make up the one and only Selection Field. The Selection Field must be organized with the default choice as the first item. The remaining choices should be arranged in priority order (i.e., most frequently used) if possible, or in logical order (i.e., in alphabetic order) when the priority is not known.

[6] When this heading refers to a required input field or when the application needs to emphasize a heading, then Level 4 will be used.

Required Panel Body Elements

1. Selection Field - two or more pairs of Key Phrase ID and

Use of the other Panel Body Elements shown is optional.

The **Interaction Technique** for this panel type is "Selection Field Interaction" on page 137.

The following are examples of Single Selection Field Menu Panels.

```
Select ONE of the following:

   1  ADD       Create a new database record.
   2  UPDATE    Modify an existing database record.
   3  DELETE    Remove an existing record from the database.

Type your selection and press Enter; otherwise press F2 (Quit).
```

Figure 89. Single Selection Field Menu Panel - Example 1

```
Select ONE of the following:

   1  ADD       Create a new database record.
   2  MODIFY    Change or delete an existing database record.

Type your selection and press Enter; otherwise press F2 (Quit).
```

Figure 90. Single Selection Field Menu Panel - Example 2

Multiple Selection Field Format: Figure 91 presents the Panel Body Elements for Multiple Selection Field Menu Panels; see Figure 83 on page 115 for location of the Common Panel Elements.

```
Top Instruction Area

   MAJOR HEADING 1   ID  Key Phrase   Explanatory Text
                     ID  Key Phrase   Explanatory Text
                     ID  Key Phrase   Explanatory Text
                     ID  Key Phrase   Explanatory Text

   MAJOR HEADING 2   ID  Key Phrase   Explanatory Text
                     ID  Key Phrase   Explanatory Text
                     ID  Key Phrase   Explanatory Text

Bottom Instruction Area
```

Figure 91. Multiple Selection Field Menu Panel - Panel Body

Selection Fields are organized with the default Key Phrase as the first item in each selection field (major group). The remaining choices should be arranged in priority order (i.e., most frequently used) if possible, or in logical order (i.e., in alphabetic order) when the priority is not known.

Required Panel Body Elements

1. Selection Field - two or more, each with two or more pairs of Key Phrase ID and Key Phrase.
2. Major Heading - one for each selection field on the panel.

Each selection field is prefixed with a letter, then numbered (see example panel in Figure 92).

Two major headings are shown with multiple choices for each; the application may allow one or more selections from major heading.

Use of the other Panel Body Elements shown is optional.

The **Interaction Technique** for this panel type is "Selection Field Interaction" on page 137.

The following are examples of Multiple Selection Field Menu Panels.

```
Select ONE option from EACH group below:

                    ID  OPTION

   TYPE OF RECORD   A1  Resource
                    A2  Location
                    A3  Vendor


   ACTION DESIRED   B1  Display
                    B2  Add
                    B3  Change
                    B4  Delete

Type each selection ID (separated by a blank), then press Enter;
otherwise press F2 (Quit).
```

Figure 92. Multiple Selection Field Menu Panel - Example 1

Entry Panels

Entry Panels require the user to input information into the dialog instead of allowing him to simply select from a list of choices. Two panel types are defined, based on the functions they perform as viewed by the user:

Parameter Entry Panels	Entering parameter data by keying the data into predefined fields ("Parameter Entry Panels").
Data Entry Panels	Entering data in a "fixed length field" ("Data Entry Panels" on page 131).

Parameter Entry Panels: This panel type gives the user a capability to input parameter information that the system requires to perform some action. If this panel type is used in conjunction with commands, then the Panel Title should be the command name and the Key Phrases should correspond to the command parameters. Only Entry Fields may be presented in this panel type.

Figure 93 on page 130 presents the Panel Body Elements for Parameter Entry Panels; see Figure 83 on page 115 for location of the Common Panel Elements.

```
Top Instruction Area

   ITEM          CHOICE         DESCRIPTION

   Key phrase. . Entry Field    Explanatory Text

   Key phrase. . Entry Field    Explanatory Text
                                Explanatory Text cont'd

   Key phrase. . Entry Field    Explanatory Text

Bottom Instruction Area
```

Figure 93. Parameter Entry Panel Body Elements

Entry Fields on Parameter Entry Panels must contain a default value, unless there is no logical way to provide one. Providing defaults allows the user to accept them without typing; the user simply executes the ENTER action.

If an Entry Field on this panel type does not contain a default value and the entry is necessary to continue the dialog, the Entry Field becomes a Required Entry Field. Required Entry Fields must be placed as one of the first fields of a panel, or one of the first fields of a logically related group of items within a panel. This minimizes cursor movement necessary within the panel.

Required Panel Body Elements

1. Key Phrase - used to describe the parameter that the user is to specify; it should be the parameter (or command) name.
2. Entry Field - used to receive the user's input. "Dot leadering" is required between the Key Phrase and the Entry Field (see "Dot Leadering" under Entry Field in "Panel Body Elements" on page 120).

The other panel body elements shown are optional.

The **Interaction Technique** for this panel type is "Entry Field Interaction" on page 137.

Figure 94 on page 131 is an example of a Parameter Entry Panel.

```
Type the desired Test Options below:

   WAIT (0-60) . . . . . --       Minutes to wait before repeating test.
   REPEAT (0-10000). . . ----     Number of times to repeat test.
   ALTERNATE (Y or N). . -        Alternate between local/remote status tests?
   SAVE (Y or N) . . . . -        Save these Test Options?

Type the desired Problem Determination (PD) Options (Y=yes, N=no):
(Note that Self Test and Channelized Tests are Disruptive.)

   MULTIPOINT (Y or N) . . -    Test related tributary modems, if needed?
   SELF TEST (Y or N). . . -    Perform local/remote self test, if needed?
   CHANNELIZED (Y or N). . -    Test channelized modems, if needed?
   SAVE (Y or N) . . . . . -    Save these PD Options?

When finished, press Enter to continue; otherwise press F2 (Quit).
```

Figure 94. Parameter Entry Panel - Example 1

Data Entry Panels: This panel type allows the user to enter data in structured field and free-key field formats.

Three panel formats are defined, based on functions they perform as viewed by the user:

Vertical	Fields are arranged one below the other in a column (Figure 95 on page 132).
Tabular	Fields are arranged one after the other on a line (Figure 96 on page 132).
Forms Fill-In	Allows entry into a "Forms" representation (Figure 97 on page 133).

The above panel formats may be combined as desired.

Required Panel Body Elements: The Entry Field is the only required element for Data Entry Panels. The other panel body elements shown in the following examples are optional, however Headings and Top Instruction Areas are strongly recommended.

The **Interaction Technique** for Data Entry Panels is "Entry Field Interaction" on page 137.

Vertical Data Entry Format: Figure 95 on page 132 presents the Panel Body Elements for Vertical Data Entry Panels; see Figure 83 on page 115 for location of the Common Panel Elements. Note that row headings are used instead of column headings.

```
Top Instruction Area

   MAJOR HEADING
      Minor Heading. . Entry Field
      Minor Heading. . Entry Field
      Minor Heading. . Entry Field

   MAJOR HEADING
      Minor Heading. . Entry Field
      Minor Heading. . Entry Field

Bottom Instruction Area
```

Figure 95. Vertical Data Entry Panel Format

This panel type is normally used for keying a single "record" at a time; the user types the data for each field, then executes the ENTER action.

Entry Fields may be grouped by having one Key Phrase act as a "major heading" for a group of Entry Fields. The Key Phrase for each Entry Field then represents a "minor heading". Major headings are set off from minor headings by indenting. Both structured and free-form Entry Fields are allowed in this panel type.

Tabular Data Entry Format: Figure 96 presents the Panel Body Elements for Tabular Data Entry Panels; see Figure 83 on page 115 for location of the Common Panel Elements.

```
 Top Instruction Area

   COLUMN HEADING     COLUMN HEADING     COLUMN HEADING

   Entry Field        Entry Field        Entry Field
   Entry Field        Entry Field        Entry Field
   Entry Field        Entry Field        Entry Field

Bottom Instruction Area
```

Figure 96. Tabular Data Entry Panel Format

This panel type may be used for keying multiple groups of fields (e.g., multiple records at a time, one per line). The user types all the data for each record and then requests the ENTER action to present all of the fields to the application.

Multiple Entry Fields per row are allowed. Both structured and free-form Entry Fields are allowed in this panel type.

Forms Fill-in Data Entry Format: Figure 97 on page 133 presents the Panel Body Elements for Forms Fill-in Data Entry Panels; see Figure 83 on page 115 for location of the Common Panel Elements.

This panel type allows fields to be formatted in a "free-form" manner. Headings are added above each individual group to distinguish the types of information requested. A variable number of entry fields may be included on a single line.

```
Top Instruction Area

   MAJOR HEADING
      Key Phrase   Entry Field      Key Phrase   Entry Field
                                    Key Phrase   Entry Field
      Key Phrase   Entry Field
      Key Phrase   Entry Field


   MAJOR HEADING
      Key Phrase   Entry Field      Key Phrase   Entry Field
                                    Key Phrase   Entry Field
      Key Phrase   Entry Field      Key Phrase   Entry Field

Bottom Instruction Area
```

Figure 97. Forms Fill-in Data Entry Panel Format

List Panels

List Panels present a list of objects to the user and allow the user to perform actions on the objects listed. This panel type consists of a list of similar data objects and an Option field for specification of the action. The Option number is a number that is associated with an action that is currently displayed on the panel; this number is typed in the command field adjacent to the object upon which the action is desired. The Option number may also be used to select an object from the displayed list for actions that are presented on a separate sequence of Menu panels. Examples of supported List Panels are located in this section.

Figure 98 presents the Panel Body Elements for List Panels; see Figure 83 on page 115 for location of the Common Panel Elements.

```
Top Instruction Area

OPTION   HEADING       HEADING       HEADING

Option   Information   Information   Information
Option   Information   Information   Information
Option   Information   Information   Information
Option   Information   Information   Information

Bottom Instruction Area
```

Figure 98. List Panel Format

Typically this panel type is used to show multiple data objects. The user can then perform one action on each object in the list. The actions may be the same or different for each object.

Required Panel Body Elements

1. Option Field - an Entry Field formatted according to the following rules:
   - The Entry Field is one character in length.
   - The Entry Field must use the standard Field Length Delimiter and other attributes of entry fields as defined in "Panel Body Elements" on page 120.

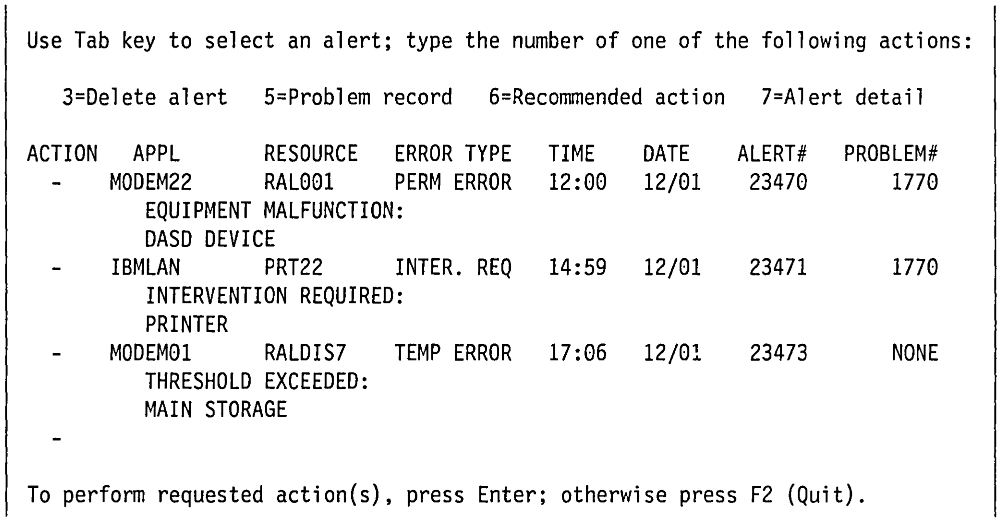
- To insure consistency across panels, the following "number=action" relationships must be used; actions not listed may be assigned other numbers.

  1=Add-like, 2=Change, 3=Delete, 4=Display

The other panel body elements shown are optional. Headings within List Panels are recommended for clarity and understanding of what is contained in each column.

The **Interaction Technique** for this panel type is "Entry Field Interaction" on page 137. Refer also to "File Management Techniques Using the List Panel" on page 155.

Figure 99 is an example of a List Panel.



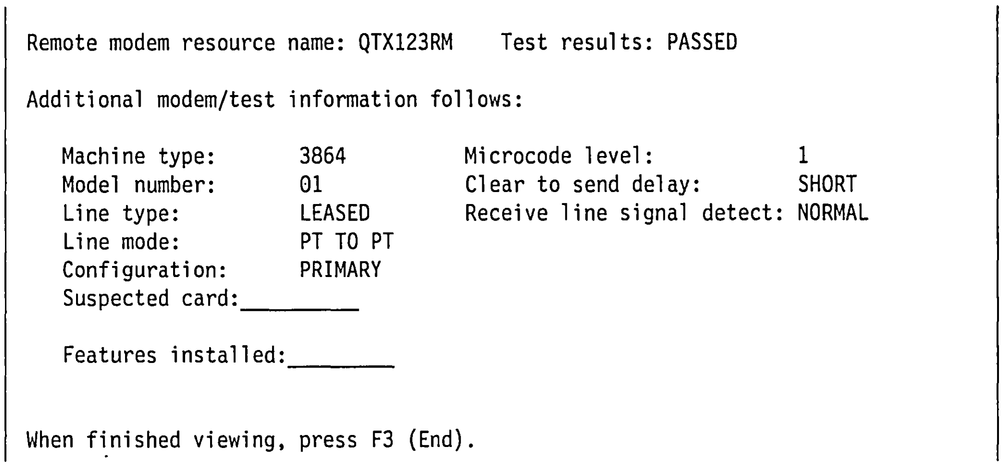
```
Use Tab key to select an alert; type the number of one of the following actions:

   3=Delete alert   5=Problem record   6=Recommended action   7=Alert detail

ACTION   APPL       RESOURCE   ERROR TYPE   TIME    DATE    ALERT#   PROBLEM#
  -     MODEM22     RAL001     PERM ERROR   12:00   12/01   23470       1770
           EQUIPMENT MALFUNCTION:
           DASD DEVICE
  -     IBMLAN      PRT22      INTER. REQ   14:59   12/01   23471       1770
           INTERVENTION REQUIRED:
           PRINTER
  -     MODEM01     RALDIS7    TEMP ERROR   17:06   12/01   23473       NONE
           THRESHOLD EXCEEDED:
           MAIN STORAGE
  -

To perform requested action(s), press Enter; otherwise press F2 (Quit).
```

Figure 99. List Panel - Example 1

Information Panels

Information Panels are used for conveying information to the user. They utilize the "Common Panel Elements" on page 115 and only contain output ("read only") data.

Figure 100 on page 135 is an example of an Information Panel.



```
Remote modem resource name: QTX123RM    Test results: PASSED

Additional modem/test information follows:

   Machine type:      3864        Microcode level:            1
   Model number:      01          Clear to send delay:        SHORT
   Line type:         LEASED      Receive line signal detect: NORMAL
   Line mode:         PT TO PT
   Configuration:     PRIMARY
   Suspected card:_________

   Features installed:_________


When finished viewing, press F3 (End).
```

Figure 100. Information Panel Example

Only Common Panel Elements are necessary in this panel type ("Common Panel Elements" on page 115); the application may use Headings, Top and Bottom Instruction Areas.

Mixing Panel Types

A mixed panel is one that contains the panel bodies of two or more of the panel types discussed previously ("Panel Types" on page 127). The panel can contain only one set of "Common Panel Elements" on page 115.

Since it is difficult to anticipate all situations where mixed panels would be necessary, there are only a few rules to follow. They are:

1. The body elements of each panel type must remain distinct. This is accomplished thru the use of White Space. For example, if a Menu Panel is mixed with a Parameter Entry Panel, the body elements of each must be visibly separate groups and they must follow the rules for their respective types (see Figure 101 on page 136).
2. Consistency of presentation must be maintained. For example, if the Top Instruction for one panel body is mixed case, the Top Instruction for the next group must also be mixed case.
3. The Menu portion of mixed panels must be presented as the bottom†most portion of the mixed panel; this allows association of the Menu panel choices with the command entry field (on line 24) into which the user types the menu choice.

Good judgement must be used when developing mixed panels. The following cautions are examples of good judgement and should be observed:

1. Utilizing more than two different panel types may become confusing to the user. Figure 102 on page 136, for example, could confuse some users. Always consider the audience being addressed. An "expert" user may prefer this complicated panel type while a novice user may be intimidated by it.
2. Top Instruction statements are highly recommended for each panel body. This technique is especially useful for the casual and novice user.
3. When mixing Information Panels with other panel types, it is appropriate to place the Information Panel above the Top Instruction for the other panel type.

This allows the user to read the information and then complete the dialog panel that follows.

Mixed Panel Examples and Guidance

Figure 101 shows an example of a Parameter Entry Panel mixed with a Menu Panel.

```
Type the name of a configuration, if known. Press F7 to see a list of names.

   CONFIGURATION NAME. . ---------


Select ONE of the following:

   1  DISPLAY   Display paths in the named configuration.
   2  DELETE    Delete the named configuration.
   3  LIST      Display names of all configurations.

Type your selection and press Enter; otherwise press F2 (Quit).
```

Figure 101. Mixed Panel Example: Entry and Menu

Figure 102 is an example of a mixture of Information, Entry and Menu.

```
Available applications are:
   ------------

   ------------



Type a record code, if necessary.

   RECORD CODE. . -   R = resource
                      L = location
                      V = vendor


Select ONE of the following:

   1  DISPLAY         Display a record.
   2  CHANGE/DELETE   Change or delete a record.
   3  ADD             Add a record.


Type your selection and press Enter; otherwise press F2 (Quit).
```

Figure 102. Mixed Panel Example: Information, Entry and Menu

Panel Dialog Management

NetView/PC Dialog Management is performed by the EZ-VU Run Time Facility (IBM Program Product 6316969) as modified for the multi-tasking environment.

Users may use the EZ-VU II Run Time Facility (IBM Program Product 6410980) for dialogue management with applications executing in the NetView/PC DOS partition.

Introduction

This section describes the facilities available to the user to carry on a dialog and includes the following sub-sections:

- "Panel Interaction Techniques,"
- "Dialog Control" on page 138,
- "Scrolling" on page 143,
- "Function Key Utilization" on page 144,
- "Messages and Prompts" on page 146,
- "Help Facility" on page 148, and
- "Color and Emphasis" on page 151.

Panel Interaction Techniques

Users interact with panels by either selecting or entering information. The techniques listed in this sub†section are divided into the categories of *Selection Field Interaction* and *Entry Field Interaction*.

Selection Field Interaction

The technique to be used for interacting with a selection field is:

- Typing a Key Phrase ID which represents the selection.

This technique allows the user to pick choices by typing the Key Phrase ID into the command entry field on the Command/Selection Line (see "Common Panel Elements" on page 115). One or more Key Phrase IDs can be entered, depending on how many selection fields are displayed and how many choices the application allows the user to pick from each field. Multiple Key Phrase IDs must be separated by one or more blanks.

When the panel is initially displayed, the command entry field contains the Key Phrase ID of the default choice(s) for the selection field(s). Multiple choices may or may not be supported by the application.

The user may elect to accept the default(s) by immediately requesting the ENTER action or the user may change the default(s) by over-typing prior to requesting the ENTER action. To simplify over-typing of defaults by the user, the application will cause the cursor to be positioned at the beginning of the field.

Entry Field Interaction

Entry fields must contain a default value unless there is no logical way to provide a meaningful one. Providing defaults allows the user to accept them by performing the ENTER action. To simplify over-typing of defaults by the user, the application will cause the cursor to be positioned at the beginning of the field.: A **Required** Entry Field is an Entry Field into which the application requires a value in order to continue the dialog and a default value is not already presented in the Entry Field.

Key Phrases of Required Entry Fields must be emphasized (see "Color and Emphasis" on page 151) to indicate to the user that an entry is required.

Dialog Control

Dialog control actions are components of the protocol that control the flow of information between a user and an application. This dialog can be viewed as sequential steps:

1. The application presents objects to the user.
2. The user requests one or more actions.
3. The application performs the action(s).
4. Repeat the above steps.

This section defines the objects and actions that are common to dialogs independent of the application being performed.

Users request dialog control actions by using techniques such as menu selections, commands, function keys, etc.

An application can provide any subset of these dialog control actions. When they are provided, they must use the terms and operate according to the rules specified in this document.

The following sub-sections will present the details of Dialog Control:

1. Dialog Control Objects

   The objects affected (characters, fields, panels, etc) by dialog control actions depend on the dialog state at a particular point in the dialog. (See "Dialog Control Objects" on page 139.)

2. Dialog Control Actions

   The fundamental dialog control actions are:

   - "ENTER" on page 139.
   - "REDISPLAY" on page 140.
   - "END" on page 141.
   - "QUIT" on page 140.
   - "MAIN MENU" on page 141.

3. Basic Dialog Control Techniques

   This section discusses Dialog Control Techniques as they relate to:

   - Function keys
   - Command Line

   See "Dialog Control Techniques" on page 142.

4. Advanced Dialog Control Techniques

   This section discusses Advanced Dialog Control Techniques such as:

   - Selection chaining
   - Typing over the Location information

   See "Advanced Dialog Control Techniques" on page 143.

Dialog Control Objects

The object affected by an action depends on the dialog state at that particular point in the dialog.

UIA (User Interface Architecture) defines **Dialog State** as the condition of the system at a particular point in the dialog as perceived by the user in terms of the objects affected (for example, characters, fields, panels, etc.).

A way to understand the change in "the object affected" that is caused by a dialog state change might be to understand the "scope" of the effect of an action.

For example, in a particular panel the user "enters" information into each field. Having completed that activity, the user "enters" the entire panel, which is the collection of all the fields. The same concept applies for panels within tasks and other hierarchical designs.

In summary, the dialog state changes when the "scope" of an action changes.

Dialog Control Actions

ENTER: All NetView/PC applications supporting an operator interface (i.e., the Dialog Manager functions) through which the user can provide input (i.e., data), must provide the ENTER action.

The ENTER action is defined as a method by which a user provides information to the application.

In other words, ENTER says to the application, "It's your turn. Now process."

The ENTER actions may be either explicit or implicit.

1. Explicit ENTER technique

   The user specifically requests the ENTER action (for example, the user presses an "enter" key). When the explicit ENTER is requested, the application checks for user input in all required entry fields. If any required entry fields are missing data, a warning message must be presented to the user and the ENTER action must not be allowed.

2. Implicit (automatic) ENTER technique

   The ENTER action can be performed automatically. For example, in many data-entry applications, a Field Advance action (Tab key) does two things:

   - causes the current field to be acted upon with respect to some kinds of validity checking, and

- causes the cursor to advance to the next field.

This automatic ENTER function is implemented through the EZ-VU field definition attribute of "panel exit".

The application's response to the ENTER action may be one or more of the following:

- Validity checking (for example, of a character in a field)
- Storing entered data as a record in a database
- Interpreting an entered number as a choice from a menu
- Presenting the next panel in the dialog

The ENTER action itself does not identify the next specific dialog state. The application must determine the next dialog state, which may or may not be based on information supplied with the ENTER action.

REDISPLAY: The REDISPLAY action discards user input within the currently displayed object (for example, a field or panel), and re-displays the object as it was when the user first saw it during the current dialog state.

We have not architected what constitutes a data commitment (SAVE). Therefore, when REDISPLAY is selected, the data that will be presented is controlled by the application.

Redisplay techniques

- If data has been placed in a panel by the user, the REDISPLAY action will discard the input that has been supplied for the current panel and the panel will be presented with its initial values.
- If the REDISPLAY action is requested while initial values are already presented, no action need take place.

QUIT: The QUIT action allows a user to make a transition ("back out") to a previously encountered dialog state without saving data.

For example, when the QUIT action is requested, the application checks for user input within the dialog. When "significant" data, as determined by the application, will be lost as a result of the QUIT action, prompting is mandatory (that is, a confirmation prompt must be presented to the user). The following example is the recommended message/prompt combination to display on the Message Line (see "Common Panel Elements" on page 115).

```
Message/prompt example:

  Do you want to save the data just entered?  (Y or N)
```

The default value for the above prompt is 'N' for no, and must be displayed in the Command/Selection field (see "Common Panel Elements" on page 115) so that the user can simply press Enter for the default value.

If the application wants to offer users a "fast path" for "QUIT and save the data", the END action can be used. See "END" on page 141.

Quit techniques

- Panel Dialog: - the QUIT action causes a transition to the prior panel presented to the user or to the previously presented panel in a higher level of panel hierarchy.

  In cases where the dialog loops (i.e., two or more panels involved in a dialog state), the application must present Bottom Instruction statements explaining how to exit the loop using the QUIT action.

- Prompt Dialog: - the QUIT action removes the prompt and the application resumes at the point in the dialog where the action was requested that caused the prompt to display.

END: The END action saves the data (if any) and takes the user to a previously encountered, application defined dialog state. This previous dialog state will be that panel from which the current dialog state was requested and entered.

When the END action is requested, the application checks for user input in all required entry fields. If data is not present for any required entry field, a warning message must be presented to the user and the END action must not be allowed.

End techniques

- Panel Dialog: - the END action causes a transition to a previously encountered, application defined dialog state. This previous dialog state will be that panel from which the current dialog state was requested and entered.

  In cases where the dialog loops (i.e., two or more panels involved in a dialog state), the application must present Bottom Instruction statements explaining how to exit the loop using the END action.

MAIN MENU: The MAIN MENU action provides a "fast path" that has the same effect as one or more END actions executed in sequence. Like the END action, the MAIN MENU action saves the data, if any, from the current panel. When a dialog is structured as a panel hierarchy, the MAIN MENU action provides a faster path out of that dialog to some previous dialog state. In the NetView/PC environment the MAIN MENU action should always take the user to the Main Menu of the application.

When the MAIN MENU action is requested, the application checks for user input in all required entry fields. If data is not present for any required entry field, a warning message must be presented to the user and the MAIN MENU action must not be allowed.

Dialog Control Action Summary: The QUIT, END and MAIN MENU are actions that cause a transition out of a portion of a dialog. Only the application (through initial program design) can determine what that dialog portion is, but an example would be a level of a panel hierarchy. Figure 103 on page 142 is presented as a visual aid in understanding "panel hierarchy" and the possible implementations of dialog control using the previously defined dialog control actions.

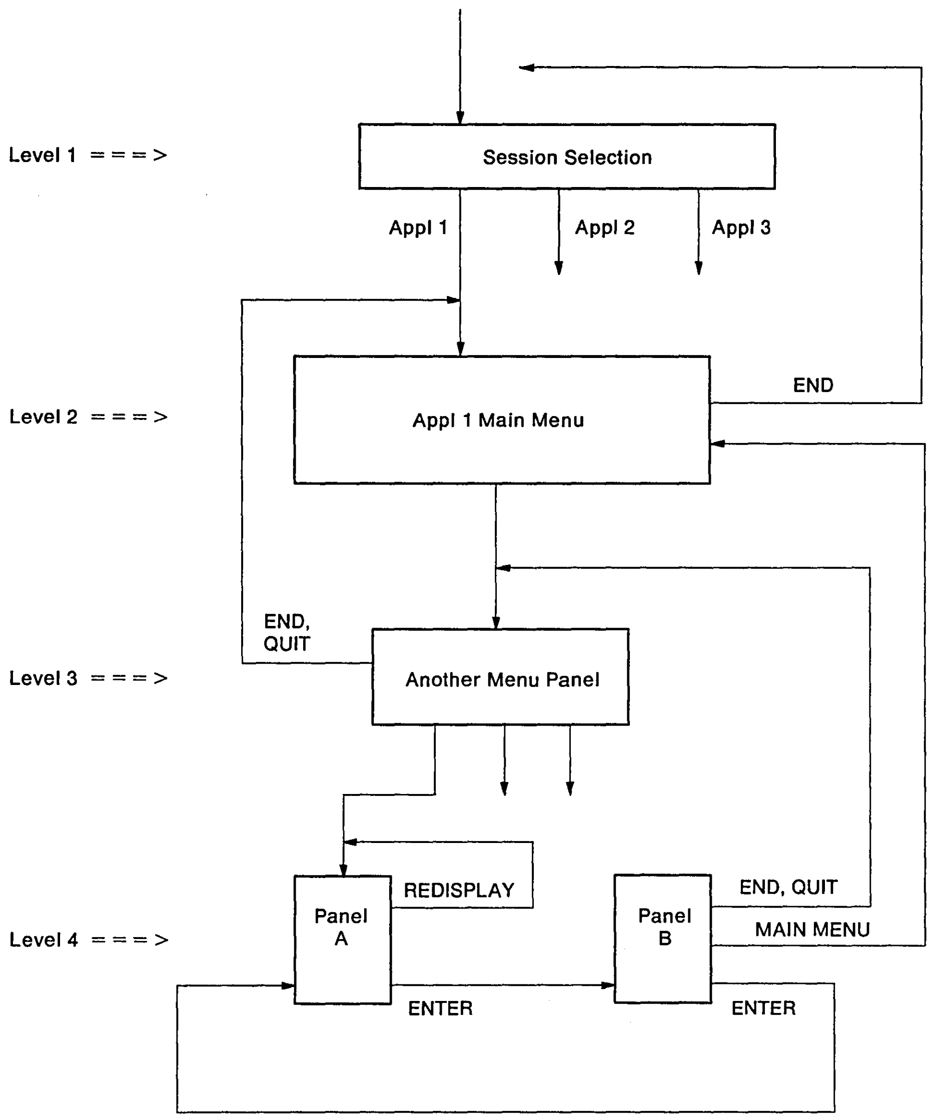


Figure 103. Portion of a Possible Panel Hierarchy (4 levels)

The boxes under "Appl 1" represent panels in which a user can enter into a dialog with NetView/PC applications. The QUIT, END and MAIN MENU actions defined previously, can be used to take the user to a particular point in a panel hierarchy. However, these actions are not allowed to take the user to a panel that he has not previously seen. Therefore the transition must always be to a panel that the user has seen earlier in the current dialog.

Dialog Control Techniques

Dialog Control Actions can be requested via function keys or via the command line. In NetView/PC dialog management, dialog control actions must be assigned to function keys (see "Function Key Utilization" on page 144). However, in addition to function keys, applications may elect to support dialog control actions via commands on the Command/Selection line.

Advanced Dialog Control Techniques

NetView/PC applications are allowed but not required to implement either or both of the following advanced techniques to support dialog control:

- "Selection Chaining."
- "Typing Over the Location Information."

Selection Chaining: Users can bypass one or more menus via this technique. The user specifies not only a selection from the menu currently presented, but also a selection from the next menu which would result from the first selection, and so on.

Users can make selections by specifying either Key Phrases or Key Phrase IDs or a mixture of both. In either case, when the user enters information, the "items" (Key Phrase, Key Phrase ID) must be separated by semicolons; this is to distinguish them from multiple choices on the Multiple Selection menus, where the choices are separated by blanks.

Example 1 - Selection Chaining by Key Phrase ID:

```
===> 3;2;5
```

In Example 1 above,

- option 3 is to be selected from the Menu Panel currently displayed, then
- option 2 is to be selected from the Menu Panel which would normally be presented next. Finally,
- option 5 is to be selected from the next Menu Panel and then the user will be presented with the panel that supports option 5 from the last Menu Panel.

Typing Over the Location Information: Another technique to cause a dialog to make a transition is to allow the Location Information element (see "Common Panel Elements" on page 115) of a panel to contain an Entry Field (see "Panel Body Elements" on page 120). The application can allow the user to type over the unprotected Entry Field in the Location Information element with the desired location information; for example, the new "page" number or the new "record" number. Suppose the Location Information element was presented as:

```
Record  19 of 213
```

If the user typed over the record number, changing it from "19" to "37", and then pressed Enter, the application would present record number 37.

Scrolling

Scrolling actions allow the user to see different portions of the information being managed by an application when there is not enough screen area to see all of it at once. When the cursor is in a scrollable information area of a panel, scrolling occurs when the cursor is moved "against" a boundary of the area. Scrolling is in the direction of cursor movement.

Techniques for two types of application scrolling are defined in this document; they are "relative" scrolling and "absolute" scrolling.

The *relative scrolling* functions are:

Function	Key Assignment and Definition
Left	For each depression of the "left arrow" key when the cursor is at the left boundary of the displayed data, the user sees the next character column to the left of the currently displayed data unless there is no more data to the left.
Right	For each depression of the "right arrow" key when the cursor is at the right boundary of the displayed data, the user sees the next character column to the right of the currently displayed data unless there is no more data to the right.
Backward (up)	For each depression of the "up arrow" key when the cursor is at the top boundary of the displayed data, the user sees the next character line above (preceding) the currently displayed data unless there is no more data above.
Forward (down)	For each depression of the "down arrow" key when the cursor is at the bottom boundary of the displayed data, the user sees the next character line below (following) the currently displayed data unless there is no more data below.
Previous	For each depression of the "PgUp" key, the user sees the "n" character lines above (preceding) the currently displayed "n" character lines.
Next	For each depression of the "PgDn" key, the user sees the "n" character lines below (following) the currently displayed "n" character lines.

The *absolute scrolling* functions are:

Function	Key Assignment and Definition
Top	When the "Home" key is depressed, the user sees the first (top) "n" character lines of information.
Bottom	When the "End" key is depressed, the user sees the last (bottom) "n" character lines of information.
Beginning of line	When the two-key combination of "Ctrl" and "left arrow" is depressed, the user sees information starting at the left-most boundary of the information.
End of line	When the two-key combination of "Ctrl" and "right arrow" is depressed, the user sees information starting at the right-most boundary of the information.

Function Key Utilization

The term "function key" refers to a key that performs a specific function; there are two kinds:

1. dedicated (hard-coded) keys, and
2. programmable (soft) keys.

Some of the dedicated keys are "Esc", "PrtSc", "PgDn", "PgUp", the directional (arrow) keys, etc. The "Enter" key is considered to be dedicated to the ENTER action.

The programmable keys, also called softkeys, are printed with "generic" labels like "F1", "F2", etc. and they may be assigned various action requests that the application allows the user to make.

In the NetView/PC environment, function keys will have a consistent assignment; the following sections describe those assignments for "hard" keys and "soft" keys.

Dedicated Function Keys

The following dedicated function keys are defined for NetView/PC:

Key	Function Definition
Enter	User request to continue and/or process input; see "ENTER" on page 139.
PgUp	User request to display previous page of information; see "Scrolling" on page 143.
PgDn	User request to display next page of information; see "Scrolling" on page 143.
Up Arrow	User request to display previous line of information; see "Scrolling" on page 143.
Down Arrow	User request to display next line of information; see "Scrolling" on page 143.
Left Arrow	User request to display next character column of information to left of currently displayed data; see "Scrolling" on page 143.
Right Arrow	User request to display next character column of information to right of currently displayed data; see "Scrolling" on page 143.
Home	User request to display the top "n" character lines of information; see "Scrolling" on page 143.
End	User request to display the bottom "n" character lines of information.

Programmable Function Keys

The following programmable function keys are defined for NetView/PC:

Key	Function Definition
F1	User request for "Help" in current dialog; see "Help Facility" on page 148.
F2	User request to "Quit" (back out) of the current dialog without saving any entered data; see "QUIT" on page 140.
F3	User request to "End" the current dialog after saving any entered data; see "END" on page 141.
F4	User request to "Return" to the application's main menu after saving any entered data; see "MAIN MENU" on page 141.
F5	User request to "Redisplay" initial values for current dialog after discarding any entered data; see "REDISPLAY" on page 140.
F6	User request to "add" an item, record, etc to a list or file; used from a List Panel.
F7	User request to see a "list" of choices for a particular input field.

Programmable function keys other than those listed above may be defined and used by NetView/PC applications.

Required Function Keys

The following function keys must be active (supported) on every NetView/PC product panel (i.e., non-help panel):

- Enter
- F1 (Help)
- Either F2 (Quit) or F3 (End)

Use of the remaining functions keys (hard or soft) is application dependent. When a key is active for a given panel, the Bottom Environment area of that panel must include the key name, and if necessary, the action assigned to that key.

Messages and Prompts

Messages and prompts are a means through which NetView/PC applications provide "feedback" to the user, and assists the user in completing a dialog with the application.

This section addresses **messages** that are issued to the user in the form of:

- Feedback that notifies the user of an error situation or incorrect action.
- Positive feedback that notifies the user that their input has been accepted and is currently being processed.
- Notification of completion of work by the application. This notification allows the user to request or initiate further actions when they must be done serially.

This section also addresses **prompts** that are used to guide the user through a dialog with an application.

- Prompts are issued as a result of an action request by the user.
- A prompt may request specific input, such as requesting the user to key some response character(s).
- A prompt may request a specific action, such as inserting a diskette necessary for continuing the application.

This section applies only to "immediate" messages and prompts that are necessary for the user to interact with the current application. Messages that are queued to a user or the workstation are displayed on the Workstation Status Line (line 25); the NetView/PC base services will manage the display of this line.

Message/Prompt Presentation

Messages and prompts are to be displayed on the Message Line (panel line 23) as described in "Common Panel Elements" on page 115.

Message/Prompt Format

Messages and prompts are displayed in mixed case in positions 5 thru 80 of line 23 of the panel. National Language requirements will necessarily limit the length of the message/prompt.

Message Types

The following usages of messages are allowed:

1. Information Messages -

   This type of message provides the user with feedback about the state of the application; typically used to tell the user that input has been accepted and is currently being or has been processed.

   Information messages that indicate a process is currently underway (for example, "Modem test running.") will be automatically removed from the display when processing is completed. Information messages may also indicate to the user that the application is busy and cannot process action requests. As a rule of thumb, anytime a process will run longer than 3 seconds, an information message must be displayed.

   Other information messages indicate that processing is complete; for example, "Record update completed."

2. Warning Messages -

   This type of message calls the user's attention to an exception condition that is not necessarily an error. For example, when large amounts of data would be lost as a result of a QUIT action, the user must be warned. For an example message, see "QUIT" on page 140. An audible alarm must be associated with this message type.

3. Action Messages -

   This type of message is used to notify the user that an improper action has been requested or that the application has had an exception condition and requires user action before the dialog can continue. An audible alarm must be associated with this message type.

Message Rules

The following rules must be observed:

1. When a dialog transaction is completed by the application and no visible result is displayed, an Information Message must be displayed that indicates completion (for example, "Record added to data base.").

2. When the application response to a user request is "delayed" due to processing, an Information Message to that effect must be issued. A "delay" is a response outside the guideline for "normal" or "average" response times for the given action request. As a general rule, anytime a process will run longer than two (2) seconds without providing some other displayed indication, then an information message must be displayed. A typical message in this situation might be "Searching, please wait...".

3. If a user request cannot be performed due to user error, each error occurrence must receive an error message. Errors should be detected as soon as constraints allow (for example, on keystroke, field, or panel level).

4. If a user requested action is partially completed by the application, an Information Message must be issued identifying what portion was completed. Additionally, the user must be instructed as to actions required for full completion of the request.

Message and Prompt Text Guidelines

The following guidelines should be used for constructing messages and prompts that are presented to the user:

1. Error messages must tell the user:
   - what is wrong, and
   - what to do to get out of the problem.
2. Messages and prompts should:
   - be complete sentences,
   - be concise but still convey a complete thought,
   - not be cryptic (avoid meaningless abbreviations/acronyms),
   - use same terminology as utilized in other parts of the application.
3. Variables inserted into messages should be enclosed in quotes; commands or actions should be in upper case for emphasis.

Help Facility

The purpose of the Help Facility is to provide operational assistance to aid the user in completing a dialog. The assistance is provided at two levels:

1. "F1" help.
2. "General" or reference help.

The Help Facility is not meant to teach; teaching is left to a Training Facility. A Training Facility is not provided with the NetView/PC product.

The following usability objectives must be observed in the Help Facility:

1. The Help Facility must enhance productivity of the user by giving the immediate information needed at a given point in the dialog.
2. The Help Facility must be available at all times.
3. The Help Facility can always be invoked with the F1 key.
4. Interaction with the Help Facility must be consistent with the interaction techniques used in interacting with any other panel within the application.
5. Help must be presented in the language of the user and should be sensitive to language issues.

The following sub-sections will further describe the Help Facility.

1. "Displaying Help Panels" on page 149
2. "Common Help Panel Elements" on page 149.
3. "Help Interaction Techniques" on page 150.
4. "Types of Help" on page 150.
5. "Content of Help Panels" on page 151.

Displaying Help Panels

In NetView/PC the F1 key is always Help; the user can press F1 at any time to request Help. The location of the cursor at the time help is requested will dictate the detail of the help information. For example, the application may support help on a "field" basis, on a "panel" basis, or both. When the cursor is in a particular field at the time help is requested and the application does not support help for that field, then help will always be presented on a "panel" basis. Somewhere within the help on a "panel" basis, the particular field help information will be given.

Utilizing multiple help panels to form a help hierarchy is to be discouraged in a Help Facility.

Common Help Panel Elements

The following are the Common Panel Elements required for all Help panels:

1. Panel Identifier (ID)

   - Purpose - Used for the referencing of a specific help panel.
   - Attributes - An alphanumeric, protected field normally eight or fewer characters in length. The Panel ID is located on line 1 of the panel, left justified in upper case.
   - Guidelines for use - A required element. The first three positions of Panel ID must be the unique component prefix assigned to the NetView/PC application; the component prefix is identified in the Product Definition File contained in the application's Distribution Diskette.

2. Help Title/Context

   - Purpose - This panel element informs the user that the information being presented is "help information" (the Title) and "to what specific subject the information refers" (the Context). For example, if help was requested from the NetView/PC Configuration Manager's Main Menu panel, the help panel might be titled:

     ```
     "CONFIGURATION HELP: MAIN MENU".
     ```

   - Attributes - An alphanumeric, protected field centered horizontally on line 1 of the panel, in upper case. The word HELP must be a portion of the title and the context must be indicated.
   - Guidelines for use - A required element.

3. Location Information

   - Purpose - If the help information requires more than one panel (page), this element is used to indicate to the user which page is currently being viewed. For example, "Page 1 of 2".
   - Attributes - An alphanumeric, protected field presented right justified on line 1 or line 2 of the help panel. Line 1 is used unless the Help Title/Context wording would extend into the Location Information.
   - Guidelines for use - Required only when multiple panels are to be presented for this particular request for help.

4. Separator

   - Purpose - Separates the Panel ID, Panel Title and Location Information from the Help Panel Body.

- Attributes - A protected field. Separation will be achieved with a solid line on line 2 of the panel. If Location Information is present and displayed on line 2, then the solid separator line will be displayed on line 3.
- Guidelines for use - A required element.

5. Help Panel Body
   - Purpose - Used for presentation of information that is dependent on the type of assistance being provided.
   - Attributes - The information in a Help panel body should be as brief as possible giving the user only what is needed to describe and/or continue a dialog with a specific object.
   - Guidelines for use - See "Types of Help" and "Content of Help Panels" on page 151.
6. Bottom Environment Area
   - Purpose - Used to inform the user on how to continue in the help dialog, including how to return to the panel from which help was initially requested. For example, the following may be presented in this area:

     ```
     F3=End   F4=Return   F6=Help Main Menu
     ```

   - Attributes - Presented in the bottom-most area of the panel, just above the Message Line.
   - Guidelines for use - A required element.

Help Interaction Techniques

Help is available to the user by pressing the F1 key from any NetView/PC product panel; help is NOT available from a "help" panel. The NetView/PC application supporting the particular help panel will use the same set of dialog control actions (see "Dialog Control" on page 138) that are used with the primary panel (that is, the panel to which the help information refers).

When help panels are displayed or removed, the Help Facility must not interfere with the current user dialog. Previously entered user data must not be destroyed and the cursor must be restored to the field it was on when help was requested. The dialog must continue as if the Help Facility was never invoked.

Types of Help

Help panels are designed to either give the user information, or to allow user interaction within the Help panel. There are two types of help panels:

Passive Help Panels: Passive Help provides information to the user. The user is allowed to read the information while continuing the dialog. If help is being provided for an "entry field", then the information should address the purpose of the field and what the user is required to enter. If help is being provided for a "protected field" such as the Data Set Name element, then the information should present a description of that field and its "value" to the user.

Interactive Help Panels: Interactive Help allows the user to carry on a dialog with the Help Facility. Such help is useful for providing Help on Help, Help on Function Keys, or Help on the Training Facility. The goal of Interactive Help is to allow the user to get information with as little disruption as possible, and then continue with the task at hand. The NetView/PC product will only provide passive help panels.

Content of Help Panels

Writers of Help Panels should keep the following in mind:

- Sentences should be complete, and concisely written.
- Help panels are meant to assist the user in progressing from one step to the next in their dialog with the application. Sentences must be action-oriented as opposed to concept-oriented. Specific steps should be used, rather than an explanation of the concept involved. Limit detail to only what is needed for the current dialog.
- When the Help Facility cannot tell what kind of information is requested, a Help Menu panel can be used to offer choices to the user.

  In these situations:

  - the first help menu panel should offer a choice of topic areas from which to choose,
  - a second help menu panel may be used to present choices for the topic area selected in the first menu, then
  - the next panel would provide the passive help information for the user.

Color and Emphasis

This section on Color and Emphasis is a required complement to the other sections of Panel Development. That is to say, NetView/PC panel definition and dialog management includes adherence to the color and emphasis requirements of this section.

Overview

Panels defined according to the rules of this section can be displayed by the NetView/PC Dialog Manager on either a color monitor or a monochrome monitor; that is to say, only one copy of the panel need be defined and the NetView/PC Dialog Manager will access an appropriate "profile" for the monitor currently in use by NetView/PC. To accomplish this, the NetView/PC Dialog Manager requires that the display attributes of panel fields be specified using the technique of "Levels of Emphasis", as defined by the EZ-VU Screen Definition Facility. Simply described, this technique is a "logical to physical" relationship. For example, if a given portion of the panel is specified to have "level 3" emphasis, then "level 3" is defined in a dynamically accessible "profile" to have two meanings: one each for color and monochrome. At "display time" the NetView/PC Dialog Manager already knows the monitor type, therefore the proper physical display attribute can be accessed from the profile and applied to the monitor for display of the current panel.

The following "special considerations" are noteworthy:

- This "Level of Emphasis" technique is applicable **only** to display of panels on a dark (black) background; that is, the physical definition of the levels will only affect the *foreground* display attributes of the monitor.
- Furthermore, certain fields on a given panel may require different display attributes during the course of dialog with the panel. For example, there are three types of messages that can be displayed (see "Message Types" on page 147), and the display attribute of the Message Line (see "Common Panel Elements" on page 115) must be different for each type. The NetView/PC Dialog Manager provides a macro interface such that a given field's display attributes can be dynamically modified for a given instance.

The remainder of this section will present various figures and charts to assist the panel designer/developer in properly assigning "level numbers" to every portion of panels being designed for the NetView/PC environment.

Figure 104 presents the IBM default display attributes for NetView/PC panels to be displayed by EZ-VU II.

LEVEL	PLACES USED	COLOR MONITOR	MONOCHROME
1	• Any field except Panel ID	Non-display	Non-display
2	• Panel ID only	Blue	Normal
3	• Data Set Name • Location Information • Separator Line • Dot Leader • Major Heading • Minor Headings • Explanatory Text • Information Field • Normal Text on Help Panels	Intensified Cyan	Normal
4	• Panel Title and Sub-title • Key Phrase ID • Heading of Required Field • Required Input Field • Command/Selection Line (text, prompt and input field) • Emphasized Text on Help Panels	Intensified White	Intensified
5	• Top Environment Area • Top Instruction Area • Bottom Instruction Area • Bottom Environment Area • Normal Text for Status Line	Intensified Cyan	Normal
6	• Normal Input Field	Intensified Green	Normal
7	• Information Text for Messages/Prompts/Status	Intensified White	Intensified
8	• Warning Text for Messages/Prompts/Status	Intensified Yellow	Intensified
9	• Action (error) text for Messages/Prompts/Status • Input field in error	Red	Intensified
10	• Reserved	Yellow	Normal
11	• Reserved	Magenta	Normal

Figure 104. NetView/PC EZ-VU II Level of Emphasis

Classes of Data

Four classes of data will be used to achieve varying degrees of color and emphasis:

1. Output - the application's presentation to the user.
2. Input - user requests to the application and user responses to the application's output.
3. Message - the application's communication to the user.
4. Status - the NetView/PC communication to the user.

Figure 105 on page 153 shows the relationship between a data class and the level of emphasis attribute. The attribute assigned to each level number is shown in Figure 104.

DATA SUB-CLASS		MEANING	LEVEL OF EMPHASIS
OUTPUT	1	• De-emphasized	Level 5
	2	• Normal	Level 3
	3	• Emphasized	Level 4
INPUT	1	• Normal	Level 6
	2	• Emphasized	Level 4
	3	• Echoed Error	Level 9[10]
MESSAGE	1	• Information	Level 7[7]
	2	• Warning	Level 8[7,8]
	3	• Action Error	Level 9[7,8]
STATUS	1	• Normal	Level 5[9]
	2	• Information	Level 7[9]
	3	• Warning	Level 8[9]
	4	• Action Error	Level 9[9]

Figure 105. Class of Data Versus Level of Emphasis Number

Output Sub-classes

1. De-emphasized Output

   This is information which one would wish to provide with less than normal emphasis.

2. Normal Output

   Normal output is used for most information "text" utilized within panels.

3. Emphasized Output

   Emphasized output is used for information that is to be "eye catching".

Figure 106 presents the Level of Emphasis assignments for the various sub-classes of output data.

[7] When more than one Level of Emphasis is required for the same field, the application program must dynamically tell EZ-VU the field attributes prior to displaying data in the field. Certain Dialog Manager macros can be used to accomplish this by specifying the panel field name and an appropriate indication of the data sub-class; subsequently the panel can be re-displayed so that the Message Line and any related input field will be displayed with the new (and appropriate) attributes.

[8] Certain data sub-classes require an audible alarm to accompany the data presentation. The Dialog Manager macros mentioned in footnote[7] can be used to accomplish this at the same time that the field attribute is changed. See also footnote[10].

[9] Status presentation (Workstation Status line 25) is managed by NetView/PC base services.

[10] In the case of Echoed Error input fields, the application must restore the field's original attributes after the input error has been resolved.

OUTPUT SUB-CLASS	PANEL ELEMENT	LEVEL OF EMPHASIS
DE-EMPHASIZED	• Top Environment Area • Bottom Environment Area • Top Instruction Area • Bottom Instruction Area	5
NORMAL	• Data Set Name • Location Information • Separator Line • Major Heading • Minor Headings – Column Heading – Row Heading – Field Heading (Key Phrase) • Explanatory Text • Dot Leader • Information Field • Normal Text on Help Panels	3
EMPHASIZED	• Panel Title and Sub-title • Key Phrase ID • Headings of Required Fields • Command/Selection Line (text and prompt) • Emphasized Text on Help Panels	4
SPECIAL	• Panel ID	2

Figure 106. Level of Emphasis Assignment for Output Data Classes

Input Sub-classes

1. Normal Input

   Normal input is for information that the user keys in dialog with the application and for information presented by the application that is modifiable by the user. It includes all entries that are not considered Emphasized Input (see next item). Examples of normal input are:

   - Optional entry fields within a panel.
   - Option column fields in a List Panel.
   - Default entries presented by the application, but are modifiable by the user.

   For more details see "Entry Field" in "Panel Body Elements" on page 120.

2. Emphasized Input

   Emphasized input is used to make a clear distinction from other panel elements. It should be used sparingly and consistently. When used for "Required Input Fields", it may become overpowering. "Required Input Fields" can fall in this sub-class or the Normal Input sub-class at the application's discretion as long as the majority of input fields on a given panel are not emphasized. Examples of emphasized input are:

   - Required entry fields.
   - Command/Selection entry field (does not include Option field on List Panels).

   For more details see "Entry Field" in "Panel Body Elements" on page 120.

3. Echoed Error

   This is used to signal invalid input from the user. The entry field in error should be re-displayed with the "Echoed Error" attribute (see Figure 105 on page 153), the cursor placed in the field and a message presented. Note that the Message Line would also need the "Action Error" attribute before displaying the message associated with the input field in error.

Figure 107 presents the Level of Emphasis assignments for the various sub-classes of input data.

INPUT SUB-CLASS	PANEL ELEMENT	LEVEL OF EMPHASIS
NORMAL	• Entry Field (displayable) • Entry Field (non-displayable)	6 1
EMPHASIZED	• Required Input Field • Command/Selection Line (input field)	6 1
ECHOED ERROR	• Input Field in Error	9

Figure 107. Level of Emphasis Assignment for Input Data Classes

Messages and Prompts Sub-classes: This sub-class is discussed in "Message Types" on page 147; please refer to that section for details on the following:

1. Information Messages
2. Warning Messages
3. Action Error Messages

Figure 108 presents the Level of Emphasis assignments for the various sub-classes of messages.

MESSAGE SUB-CLASS	MESSAGE TEXT	LEVEL OF EMPHASIS
INFORMATION MESSAGE	Information Text	7
WARNING MESSAGE	Warning Text	8
ACTION ERROR MESSAGE	Error Text	9

Figure 108. Level of Emphasis Assignment for Message Sub-classes

Status Sub-classes: Status sub-class presentation details for Workstation Status Line (line 25) is managed by NetView/PC base services. Figure 109 presents the Level of Emphasis assignments for the various sub-classes of status text.

STATUS SUB-CLASS	STATUS TEXT	LEVEL OF EMPHASIS
NORMAL STATUS	Normal Status Text	5
INFORMATION STATUS	Information Status Text	7
WARNING STATUS	Warning Status Text	8
ACTION STATUS	Action Status Text	9

Figure 109. Level of Emphasis Assignment for Status Text Sub-classes

File Management Techniques Using the List Panel

This section describes the approved dialog management techniques when using List Panels (see "List Panels" on page 133) to provide file maintenance support of data files. An assumed scenario is used for the purpose of providing an overview of following sections which present the information in a more detailed manner.

To begin the scenario, assume a panel hierarchy as presented in Figure 110 on page 157. The Maintenance Menu Panel provides access to two basic functions: add items to a list and modify items in a list. When the "MODIFY" option is selected from the Menu Panel, the application determines if the List Panel would be empty and if so, presents a message on the Menu Panel. At this point the operator would select the "ADD" option to create new records for the list.

On the List Panel the operator is allowed to specify one of several action codes for each item in the list (the example in this scenario is limited to three action codes). The application's sequence of processing the action codes should be deletes, then changes and any other actions and finally add-likes. If the list is scrollable (i.e., more items in the list than can be presented on one display), the application must handle the action codes entered over the entire range of items, i.e. multiple pages. For example, the operator may enter actions codes on a certain page, then instead of pressing Enter to perform those actions, the operator can press PgUp or PgDn to see other pages of items and enter action codes on those items before finally pressing Enter to process all of the action codes entered throughout the file of data.

For delete actions, the application must determine if significant data loss would occur by deleting any one of the indicated items. If deemed not significant, the delete(s) can occur immediately. If deemed significant or indeterminable, the application must issue a request for confirmation of the delete request(s). Only one confirmation is required for all items to be deleted. Items are deleted from the primary storage location (presumed to be a disk file). The deleted item's position on the List Panel is NOT to be removed; use a notation at the item location to indicate that the item has been deleted.

For change actions, the indicated item is fully presented on the Change Panel. The operator has the option to change certain data (as determined by the application) or not to change any data. Changed items are immediately updated in the primary storage location of the items (presumed to be a disk file). The changed portion of the item is NOT to be reflected on the List Panel; use a notation at the item location to indicate that the item has been changed.

For add-like actions, the application will present the Add panel initialized with the values of the item indicated. This is in effect an add action with a specified "prototype" and is useful when the new record is to be similar to an existing record. The operator can Quit (return to the invoking panel) at any time or can continue creating new records by alternately pressing Enter to add a new record, then change necessary fields and press Enter again to add another record. Newly created records are NOT to be reflected on the currently displayed List Panel.

If the application supports a function to add items to the list (most applications will), then the primary invocation of the add function will be from a Maintenance Menu Panel directly to the Add panel. The application's List Panel should support a fast-path to the Add panel via a function key (F6=Add), in addition to the recommended "add-like" action code which is a "prototype add".

It is recommended that the List Panel be "time-stamped", i.e. place the current time on line 1 right-justified (above any location information). If location information requires this space, then the time stamp can be placed right-justified below the separator line (below the location information). This time-stamp reflects the chronological status of the items currently being displayed on the List Panel. The operator should be free to choose when the displayed list is to be "refreshed". The Redisplay function (F5=Redisplay) is used to accomplish the list refreshing. The "refreshed" List Panel will have a new timestamp, deleted records are no longer remembered, new values of changed records are displayed along with all newly added records. The application must decide which page of the new list is to be initially presented (i.e., default to page 1 or some calculation based on items presented at time of Redisplay request).

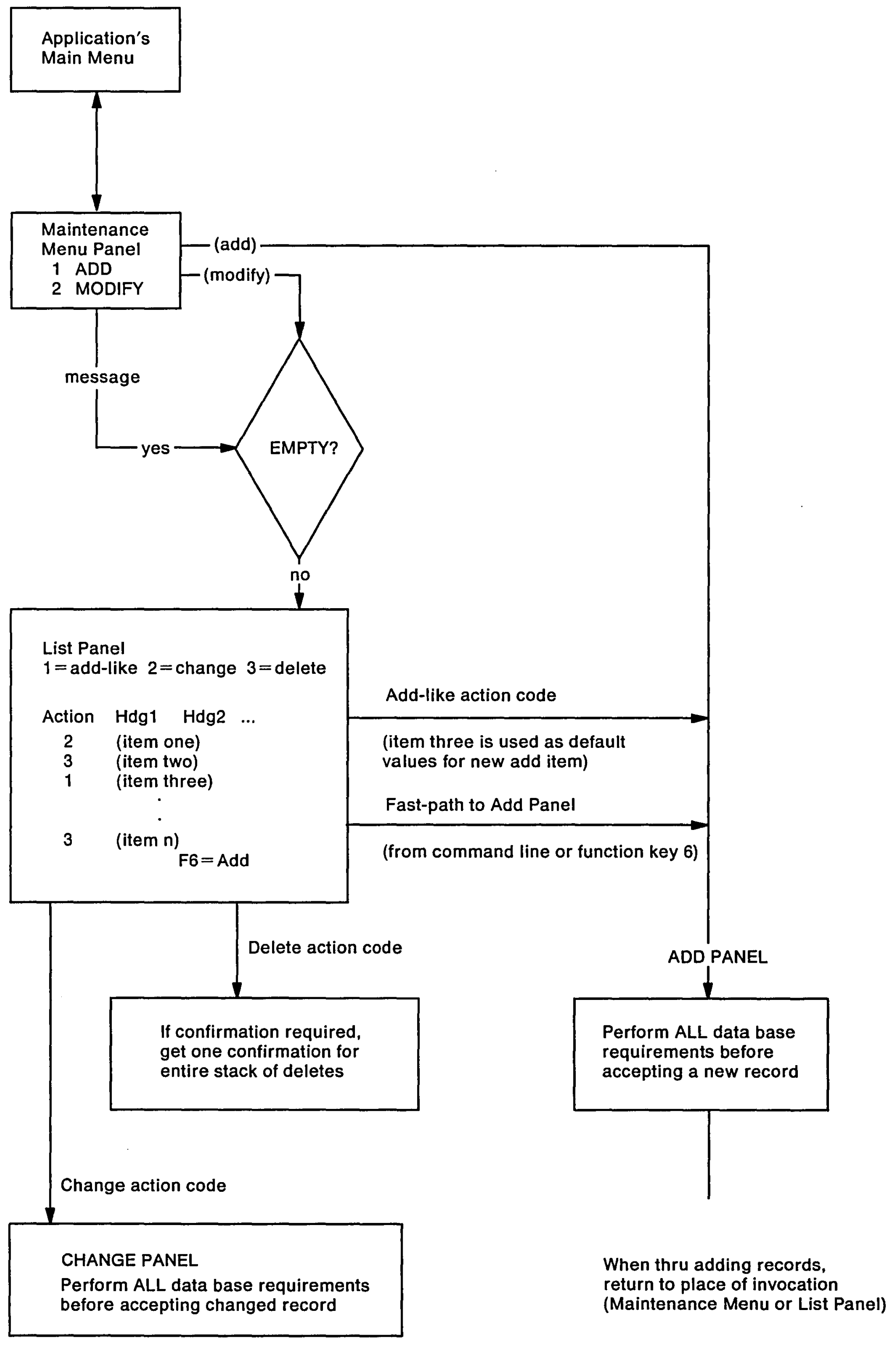

Application's Main Menu
Maintenance Menu Panel
1 ADD
2 MODIFY
(add)
(modify)
message
yes
EMPTY?
no
List Panel
1=add-like 2=change 3=delete
Action Hdg1 Hdg2 ...
2 (item one)
3 (item two)
1 (item three)
3 (item n)
F6=Add
Add-like action code
(item three is used as default values for new add item)
Fast-path to Add Panel
(from command line or function key 6)
Delete action code
ADD PANEL
If confirmation required, get one confirmation for entire stack of deletes
Perform ALL data base requirements before accepting a new record
Change action code
CHANGE PANEL
Perform ALL data base requirements before accepting changed record
When thru adding records, return to place of invocation (Maintenance Menu or List Panel)

Figure 110. Panel Hierarchy Example

Maintenance Menu Panel

This type of panel would be displayed to give the operator the following options:

1. add items to a new or existing file, or
2. modify items within an existing file.

For the situations which require only the addition of items to a file, the "add" option allows the operator to bypass the potential performance overhead associated with the "modify" option. When the "add" option is requested, the application displays the Add Panel and allows the actions described in the section titled "Add Panel Dialog Control" on page 160.

The "modify" option is provided so that the operator can selectively apply a wide range of actions to the items contained within the file. Those actions will traditionally include the ability to change and delete items within the file as well as the ability to add new items to the file. Additionally the application may allow other actions on items within the file or associated with the file.

When the "modify" option is requested and the related data file does not exist, an information message is presented on the Maintenance Menu panel indicating that the file is empty. When the data file does exist the application will display the List Panel and allow certain pre-determined actions to be specified on the items in the file. Action codes generally allowed will include "change", "delete" and possibly "add-like". Also a "fast path" to the Add Panel may optionally be supported using function key 6 (Add). Section "List Panel" will discuss these action codes.

List Panel

The application accesses the data file associated with the List Panel and displays the file items on the List Panel in a pre-determined order. The operator is allowed to type one action code (request) per item; all items not currently displayed may be viewed thru scrolling actions and those items may also be assigned action codes.

When the operator presses Enter, the application reviews all items in the list for action codes; the sum total of action codes becoming what is called the "stack" of action requests. When the "stack" becomes "empty" or when the "stack" is terminated prematurely by the operator, the application will display a "summary message" of the action(s) completed (e.g., nnn of nnn deletes; nnn of nnn changes; nnn adds processed"). This message will appear on either the List Panel, the Maintenance Menu Panel or the application's Main Menu panel depending upon how the stack is completed or terminated.

Delete Function

The "delete function" to delete an existing item from a primary data file is invoked by typing the "delete" code next to any item on the List Panel and then pressing the Enter key. This procedure will cause the application to delete each indicated item from the primary data file. The application has the responsibility to have predetermined if the deletion(s) will cause significant loss of data; if so the application will present an "Are you sure?" message on the List Panel and a prompt "(Y/N)" for operator response. If a "N" is entered, the delete request(s) are canceled. If a "Y" is entered, the delete function will delete from the primary data file all records that are currently noted with the delete action code.

List Panel Dialog Control

From the List Panel the following actions are allowed:

- Help
  - help information is presented
- Enter
  - create a stack of requests from the action code(s) typed on items in the list,
  - perform the requests in the order of:
    1. delete requests (see "Delete Function" on page 158),
    2. change requests (see "Change Function"),
    3. add-like requests (see "Add Function" on page 160).
  - present a summary message on the List Panel,
    e.g., "nnn of nnn deletes, nnn of nnn changes, nnn adds processed."
- Quit
  - all action codes are ignored; the primary data file is NOT modified in any way,
  - the previous panel is displayed (i.e., the Maintenance Menu Panel).
- Main Menu
  - the same action as "Quit" except that the next panel to be displayed is always the application's Main Menu panel.
- Redisplay
  - *all* action codes are ignored,
  - the List Panel is "re-freshed" with the current data on the primary data file,
  - the current time is reflected on the List Panel.
- Add -
  - a "fast-path" to the Add Panel (see "Add Function" on page 160).

Change Function

The "change function" to change an existing item on a primary data file is invoked by typing the "change" code next to any item on the List Panel and then pressing the Enter key. This procedure will cause the application to display the Change Panel; see the section titled "Change Panel Dialog Control."

Change Panel Dialog Control

The application displays the Change Panel with the values of the first change item on the stack of requests.

From the Change Panel the following actions are allowed:

- Help
  - help information is presented
- Enter
  - the item on the primary file is updated (unless no data was changed),
  - "trace notation" for the List Panel is generated for this changed item,

  - if there is another change request in the stack:
    - display the Change Panel with the values of the next item that was marked with the change action code,
    - present on the Change Panel, a "change status" message about the *previous* change action (e.g., "Previous change completed").
  - if there are no more change requests in the stack, return to the application for further processing of the stack (i.e. add-like requests).
- Quit
  - the current change request is not processed,
  - *all* remaining requests in the stack are flushed (ignored),
  - "trace notation" for the List Panel is generated only for change requests processed prior to this Quit action,
  - return to the application to display the List Panel with a summary message of actions completed.
- End
  - same as "Quit" except that current change request is processed (primary file updated).
- Main Menu
  - same as "End" except that next panel to be displayed is the application's Main Menu.

Add Function

The "add function" to add items to a primary data file can be invoked in any of the following ways:

1. by selecting "Add" from the Maintenance Menu Panel, or
2. by pressing function key 6 on the List Panel, or
3. by typing the "add-like" code next to any item on the List Panel and then pressing the Enter key.

Either of the above steps will cause the application to display the Add Panel; see the section titled "Add Panel Dialog Control."

Add Panel Dialog Control

The application displays the Add Panel and the default values displayed are as follows:

- application-determined values when the add request is from the Maintenance Menu panel or the "F6=Add" action from the List Panel,
- actual values from another list item are used when the add request is from an "add-like" action code of the List Panel.

From the Add Panel the following actions are allowed:

- Help
  - help information is presented

- Enter
  - new data is saved in the primary data file,
  - if the stack is empty, display the Add Panel using the values of the most recently added item as default values; the panel will contain a confirmation message about the new add.
  - if the stack is not empty, display the Add Panel using the values of the indicated item as default values; the panel will contain a confirmation message about the previous add.
- Quit
  - the current add request is not processed,
  - *all* remaining requests in the stack are flushed (ignored),
  - return to the application to display the next panel with a summary message of actions completed; the next panel will be the panel from which the Add function was requested (Maintenance Menu panel or List Panel).
- End
  - same as "Quit" except that any new data on the panel is saved before displaying the next panel.
- Main Menu
  - same as "End" except panel to be displayed is the application's Main Menu panel.

Part 4. Sample Programs

Appendix F. DOS Sample Program Planning and Installation 167
Prerequisites 167
Components 167
  Code 167
  Panels 168
  Sample NMVTs 168
  Sample SPCF Build ARBs 169
  Sample NetView Command Processors 169
  Other 169
Installation 170
Assembly and Link 170
  Assembling the Sample Program 171
  Linking the Sample Program 171

Appendix G. Operation 173
Startup 173
General Information 173
  Function Keys 174
  File Formats 174
    ARB File Format 174
    NMVT File 175
  Other Information 175
    Output Fields 175
    Error Messages 175
Main Menu 176
  Function Keys 176
  Parameters 176
  Output Fields 177
  Shutdown 177
Alert Interface Panel 177
  Function Keys 178
  Parameters 178
  Output Fields 179
Operator Communications Interface 179
  Function Keys 180
  Parameters 180
  Output Fields 180
Service Point Command Facility Interface 181
  Function Keys 181
  Parameters 181
  Output Fields 182
SPCF Run Command 182
  Function Keys 183
  Parameters 183
  Output Fields 185
SPCF Receive Unparsed Command 185
  Function Keys 186
  Parameters 186
  Output Fields 187
SPCF Parse Host Command 187
  Function Keys 188
  Parameters 188

Output Fields 188
SPCF Build Response 189
Function Keys 189
Parameters 189
Output Fields 190
SPCF Display File 190
Function Keys 191
Parameters 191
Output Fields 192
SPCF Unformatted Display Panel 192
Function Keys 192
Parameters 193
Output Fields 193
SPCF Display Run Command 194
Function Keys 194
Parameters 195
Output Fields 195
SPCF Display Link PD Command 195
Function Keys 196
Parameters 196
Output Fields 196
SPCF Display Link Data Command 197
Function Keys 197
Parameters 198
Output Fields 198
SPCF Display Link Test Command 198
Function Keys 199
Parameters 199
Output Fields 199
SPCF Send Unformatted Response 200
Function Keys 200
Parameters 201
Output Fields 202
SPCF Send Unsolicited Message 202
Function Keys 203
Parameters 203
Output Fields 204
SPCF Send Error Sense Data 204
Function Keys 205
Parameters 205
Output Fields 206
SPCF Build Message 207
Function Keys 207
Parameters 207
Output Fields 207
SPCF Correlator List 208
Function Keys 209
Parameters 209
Output Fields 209
Host Data Facility Interface 209
Function Keys 210
Parameters 210
Output Fields 211

Appendix H. API Sample Program Error Messages 213

Appendix I. DOS Sample Program Code 223
APIMAIN.DSG 223
APIMAIN.MAC 242
APIMAIN.EXR 245
APIMAIN.DEF 246
APIMAIN.ASM 247
APIMAIN.UTL 292
APIUTIL.DSG 303
APIUTIL.EXR 310
APIUTIL.ASM 311
APIDISP.DSG 332
APIDISP.ASM 340

Appendix J. NetView Sample Programs 361
NetView Sample Presentation Services Command Processor (PSCP) 361
NetView Sample Data Services Command Processor (DSCP) 371

Appendix F. DOS Sample Program Planning and Installation

Prerequisites

The API Sample Program requires the following software in order to run:

1. NetView/PC 1.1.
2. EZ-VU II Runtime Facility.

In addition, if a new executable file is to be generated from the provided source code or object files, the following additional software is required:

1. IBM Macro Assembler Version 2.0.

Components

Code

The API Sample Program consists of the following files:

Filename	Contents
APIMAIN.EXE	Executable code file.
APIPANEL.LIB	Library file containing panels.
APIMAIN.ASM	Source code for major portion of Sample Program.
APIMAIN.DSG	Declarations of variables used in APIMAIN.ASM.
APIMAIN.DEF	Definitions of constants.
APIMAIN.MAC	Macro definitions.
APIMAIN.UTL	Special purpose utility procedures used in APIMAIN.ASM.
APIMAIN.EXR	File containing ″extern″ statements for public entities in APIMAIN.
APIUTIL.ASM	General purpose utility routines.
APIUTIL.DSG	Declarations of variables used in APIUTIL.ASM.
APIUTIL.EXR	File containing ″extern″ statements for public entities in APIUTIL.
APIDISP.ASM	Display procedures.
APIDISP.DSG	Declarations of variables used in APIDISP.ASM.

Panels

The API Sample Program uses the panel library APIPANEL.LIB. This library contains the following panels:

Filename	Panel Description
DCJVAP00.PAN	Alert API Panel.
DCJVBP01.PAN	SPCF Parse Host Command Panel.
DCJVBP02.PAN	SPCF Build Response Panel.
DCJVBP03.PAN	SPCF Unformatted Display Panel.
DCJVBP04.PAN	SPCF Formatted Display Panel.
DCJVCP00.PAN	SPCF API Panel.
DCJVCP01.PAN	SPCF Run Command Panel.
DCJVCP02.PAN	SPCF Receive Unparsed Command Panel.
DCJVCP03.PAN	SPCF Send Unformatted Response Panel.
DCJVCP04.PAN	SPCF Send Message Panel.
DCJVCP05.PAN	SPCF Send Error Panel.
DCJVCP06.PAN	SPCF Build Message Panel.
DCJVCP07.PAN	SPCF Correlator Selection Panel.
DCJVCX00.PAN	Display File Pop-up Panel.
DCJVDP00.PAN	Host Data Facility API Panel.
DCJVMP00.PAN	API Sample Program Main Menu Panel.
DCJVOP00.PAN	Operator Communications API Panel.
ISPFMNT1.PAN	EZ-VU II Runtime Facility panel.

Sample NMVTs

Filename	Contents
GOODNMVT.NMV	Contains the binary image of a sample non-generic alert.
GENERIC.NMV	Contains the binary image of a sample generic alert.
HYBRID.NMV	Contains the binary image of a sample hybrid alert.

Sample SPCF Build ARBs

Filename	Contents
RUNCMD.RSP	Contains the binary image of a sample SPCF Build ARB and its associated data. Enter this filename as the 'ARB Input Filename' on panel DCJBP02 in Figure 118 on page 189 in order to build a response to a RUNCMD command.
LINKPD.RSP	Contains the binary image of a sample SPCF Build ARB and its associated data. Enter this filename as the 'ARB Input Filename' on panel DCJBP02 in Figure 118 on page 189 in order to build a response to a LINKPD command.
LINKTEST.RSP	Contains the binary image of a sample SPCF Build ARB and its associated data. Enter this filename as the 'ARB Input Filename' on panel DCJBP02 in Figure 118 on page 189 in order to build a response to a LINKTEST command.

Sample NetView Command Processors

Filename	Contents
NVPCDSCP.BAL	Contains 370 assembler language source for a sample NetView Data Services Command Processor (DSCP).
NVPCPSCP.BAL	Contains 370 assembler language source for a sample NetView Presentation Services Command Processor (PSCP).

Other

In addition to the above, various other files are also included:

Filename	Contents
README.API	Contains information regarding the API Sample Program.
INSTALL.BAT	Installs the API Sample Program onto a fixed disk.
SAMPLE.BAT	Executes the API Sample Program.
MASMSAMP.BAT	Assembles the API Sample Program source code modules.
LINKSAMP.BAT	Links the API Sample Program object files.
VAPI.MSG	Messages (error and otherwise) displayed by the API Sample Program.

Installation

The following steps should be taken in order to install the API Sample Program onto a fixed disk.

Note: NetView/PC must be installed onto a fixed disk before the API Sample Program is installed.

1. Insert the diskette containing the API Sample Program into the desired floppy drive.
2. Type:

   ```
   a: install c:
   ```

   where *a* and *c* are the drive letters of the source and destination drives, respectively.

This will copy the panels into the \NVPCPANL directory, and other files into the \NETVIEW directory.

Note: There must be at least 700K of free space on the fixed disk for the installation to complete successfully.

The installation file will indicate whether the installation was completed successfully or not.

Assembly and Link

In order to assemble and link the API Sample Program, the following conditions must be satisfied:

Note: The following steps need not be performed to obtain a runable copy of the API Sample Program; an executable file is distributed with the sample program.

1. NetView/PC and the API Sample Program must be installed on a fixed disk.
2. The current directory must be **\NETVIEW**.
3. ISPASM.OBJ (from the EZ-VU runtime facility) must be present in the current directory.
4. MASM V2.0 must be present in the search path.
5. LINK (the DOS linker) must be present in the search path.
6. Enough space must be present on the fixed disk for the output files.
7. NetView/PC should not be running.

Warning: Attempting to assemble or link API Sample Program while NetView/PC is running may lead to system failure. Assemble and link only under DOS.

Assembling the Sample Program

In order to assemble the components of the API Sample Program:

1. Type `MASMSAMP` on the command line.

This will run the macro assembler, and produce the object and listing files for APIMAIN, APIDISP, and APIUTIL. If no listing file is desired, the batch file **MASMSAMP.BAT** may be edited in order to remove the listing filename from the MASM invocation line.

Linking the Sample Program

In order to link the components of the API Sample Program:

1. Type `LINKSAMP` on the command line.

This will run the DOS linker and produce the **APIMAIN.EXE** file, as well as a link map in **APIMAIN.MAP**.

Appendix G. Operation

Startup

The following steps should be taken in order to run the sample program.

Note: The instructions below assume that NetView/PC and the API Sample Program have been installed on the **c:** drive.

1. Set the default drive to **c:** by typing `c:` on at the DOS command prompt.
2. Type `cd \NETVIEW` to set the default directory to **\NETVIEW**.
3. Start NetView/PC with a configuration including Alerts, Host Data Facility, SPCF, and a DOS partition with a size of at least 240K (e.g. `netview` *`groupfilename`*) where *groupfilename* is a grouping file created using the NetView/PC operator services grouping function.

   Note: Only the facilities which are to be tested need be placed in a group; i.e. if only the SPCF portion of the API Sample Program is to be exercised, only SPCF needs to be included in the grouping file. This sample program will not execute with the grouping STARTDP.
4. Select the DOS partition from the NetView/PC Session Selection Menu.
5. Start the sample program by typing `sample`.

General Information

The operation of the API Sample Program is described in detail, panel by panel, in the following sections:

- "Main Menu" on page 176.
- "Alert Interface Panel" on page 177.
- "Operator Communications Interface" on page 179.
- "Host Data Facility Interface" on page 209.
- "Service Point Command Facility Interface" on page 181.
- "SPCF Parse Host Command" on page 187.
- "SPCF Build Response" on page 189.
- "SPCF Run Command" on page 182.
- "SPCF Receive Unparsed Command" on page 185.
- "SPCF Send Unformatted Response" on page 200.
- "SPCF Send Unsolicited Message" on page 202.
- "SPCF Send Error Sense Data" on page 204.
- "SPCF Build Message" on page 207.
- "SPCF Correlator List" on page 208.
- "SPCF Display File" on page 190.
- "SPCF Unformatted Display Panel" on page 192.
- "SPCF Display Link Data Command" on page 197.
- "SPCF Display Link PD Command" on page 195.

- "SPCF Display Run Command" on page 194.
- "SPCF Display Link Test Command" on page 198.

General features of the API Sample Program are described in these sections:

- "Function Keys" describes the functions keys common to all the API Sample Program panels.
- "File Formats" describes the format of the files used by the API Sample Program.
- "Other Information" on page 175 provides other general information.

Function Keys

While the active function keys vary from panel to panel in the API Sample Program, some are constant across panels. The most common keys are listed below (note that not all appear on all panels):

Key	**Description**
F1	Displays a help message.
F3	Returns to the previous panel, or to the DOS partition if pressed on the main menu. Note that any changes made to the input fields on the panel will be saved.
F6	Invokes a secondary copy of the DOS command processor.
F9	Displays a file.
F10	Call the NetView/PC API.

File Formats

ARB File Format

The ARB begins the file, stored bit for bit as it would be in memory with one exception: pointers represent byte displacements from the start of the file rather than being addresses of data in memory. The sample program only accepts files less than 64K in length, so for all 32 bit pointers (Intel DD format), the segment portion will be zero, and the offset word will be the offset into the file.

Following the ARB are 10 bytes for the Correlator, Send or Receive. These may be set to zero to indicate absence of a valid correlator, but the Correlator field is required even if it is all zeros. Following the correlator field are any additional data required by the ARB.

For example consider the file format for a parsed RUN command:

- ARB -- length 35 -- displacement 0

  In the ARB is a pointer to the command; in this case the value is 45, in Intel Format: 2D000000H.
  The bytes are reversed as are the displacement and segment. In other words, the leftmost byte is least significant and the bytes to the right are more significant.

- Receive Correlator -- length 10 -- displacement 35

- Command text -- length variable up to 255 -- displacement 45. The length is in the ARB.

NMVT File

The NMVT begins the file, stored bit for bit as it would be in memory.

Following the NMVT are 10 bytes for the Correlator, Send or Receive. These may be set to zero to indicate absence of a valid correlator. The Correlator field is required even if it is all zeros.

Other Information

Output Fields

One of the sections included in the description of each panel is the "Output Fields" section. This describes the fields on the panel which display data returned by the sample program when an action is taken.

Some of these output fields are present on more than one panel, and so are described only once, below. These include:

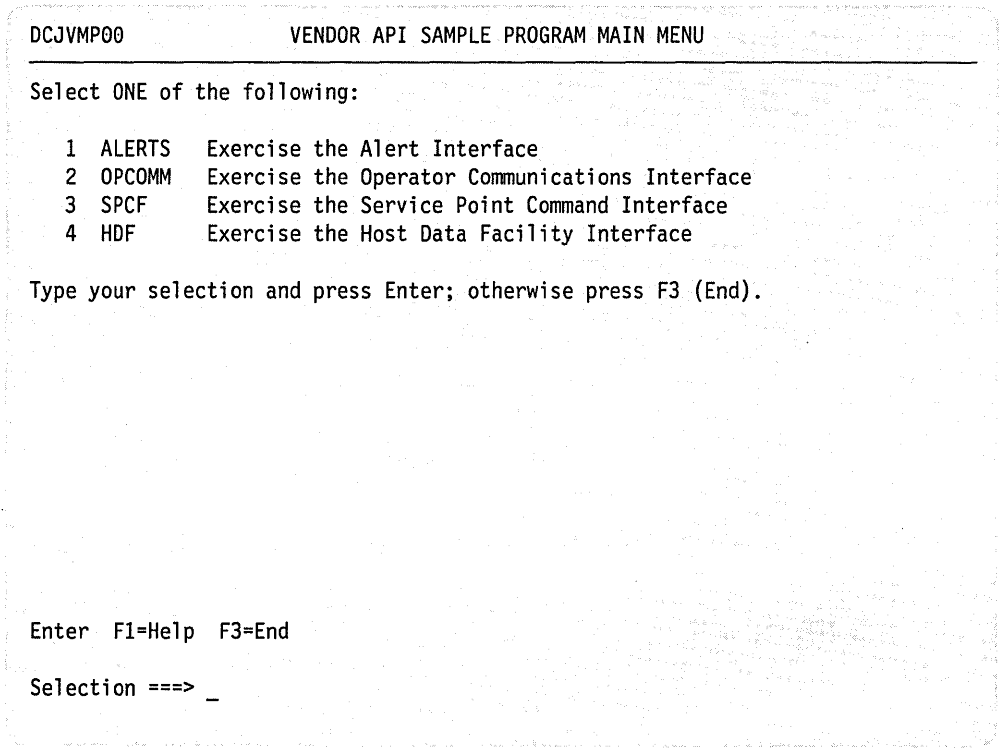
Field	**Description**
ARB ID Found	This field is present on all panels where a call to the API may be made. Its value indicates whether the NetView/PC API routine found a valid ARB ID in the ARB which it received.
Return Code	This field is present on panels which allow calls to the API subroutines. It corresponds to the return code field in the ARB.
Error Class	This field is present on panels which allow calls to the API subroutines. It corresponds to the Error Class field in the ARB.
Error Type	This field is present on panels which allow calls to the API subroutines. It corresponds to the Error Type field in the ARB.

Note: The values of the Return Code, Error Class, and Error Type are dependent on the ARB in which they are found. Refer to the main body of this document for information regarding the meanings of returned values.

Error Messages

When errors are detected by the API Sample Program, an error message is placed on the display. These error messages are documented in Appendix H, "API Sample Program Error Messages" on page 213. Note that errors returned by the API Sample Program are distinct from errors returned by the NetView/PC API subroutines which the sample program calls.

Main Menu



```
DCJVMP00             VENDOR API SAMPLE PROGRAM MAIN MENU
─────────────────────────────────────────────────────────────────
Select ONE of the following:

   1  ALERTS   Exercise the Alert Interface
   2  OPCOMM   Exercise the Operator Communications Interface
   3  SPCF     Exercise the Service Point Command Interface
   4  HDF      Exercise the Host Data Facility Interface

Type your selection and press Enter; otherwise press F3 (End).










Enter  F1=Help  F3=End

Selection ===> _
```

Figure 111. Main Menu

The API Sample Program main menu is the first panel displayed when the API Sample Program is started. It is used to select the API which is to be exercised.

Function Keys

There are no additional active function keys, aside from those described in "Function Keys" on page 174.

Parameters

The only entry parameter on this panel is:

Selection = *n*

n is a number from one to four corresponding to the number of the desired function on the main menu (Figure 111). The functions presently available are:

ALERTS	Alert API (see "Alert Interface Panel" on page 177).
OPCOMM	Operator Communications API (see "Operator Communications Interface" on page 179).
SPCF	Service Point Command Facility API (see "Service Point Command Facility Interface" on page 181).
HDF	Host Data Facility API (see "Host Data Facility Interface" on page 209).

Output Fields

There are no output fields on this panel.

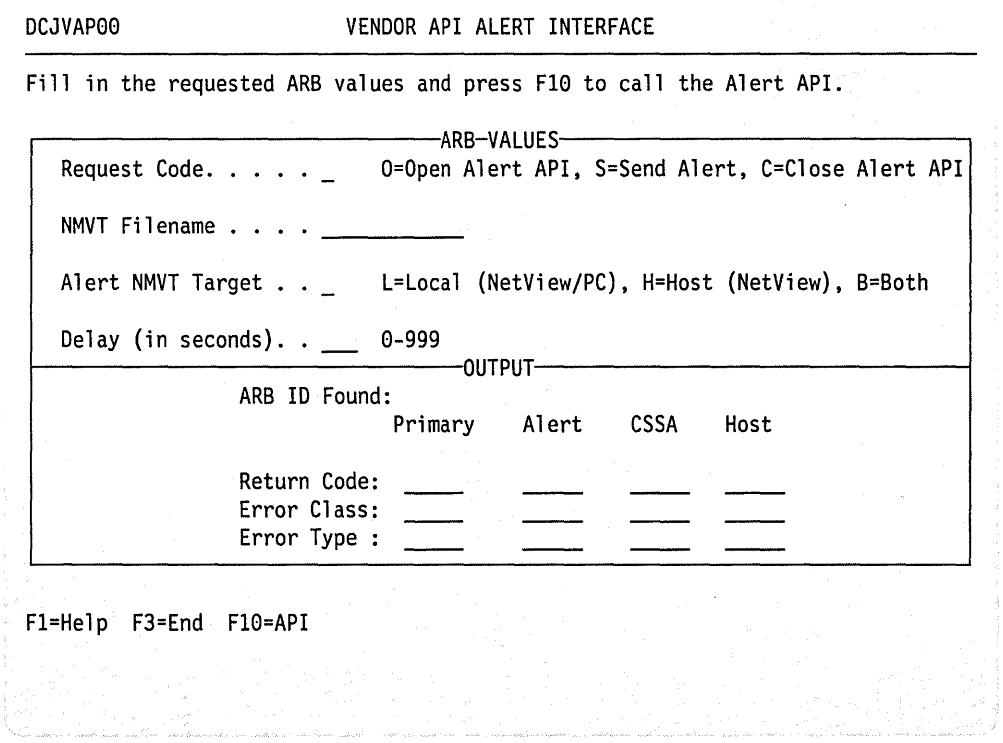
Shutdown

In order to exit the API Sample Program, press F3 while on the main menu. This will cause the API Sample Program to relinquish control back to the DOS partition.

Warning:

1. The API Sample Program will allow you to exit back to the DOS partition without performing the required shutdown of any API interfaces you may have opened.
2. Terminating the program without closing all open interfaces results in a loss of resources that cannot be reclaimed without shutting down NetView/PC and bringing it back up.
3. You should therefore be certain that you have closed all the interfaces you opened before you shut down the API Sample Program, (unless you wish to create a loss of resources for test purposes).

Alert Interface Panel



```
DCJVAP00                    VENDOR API ALERT INTERFACE
------------------------------------------------------------------------------
Fill in the requested ARB values and press F10 to call the Alert API.

+--------------------------------ARB-VALUES----------------------------------+
| Request Code. . . . . _   O=Open Alert API, S=Send Alert, C=Close Alert API|
|                                                                            |
| NMVT Filename . . . . ____________                                         |
|                                                                            |
| Alert NMVT Target . . _   L=Local (NetView/PC), H=Host (NetView), B=Both   |
|                                                                            |
| Delay (in seconds). . ___ 0-999                                            |
+----------------------------------OUTPUT------------------------------------+
|                 ARB ID Found:                                              |
|                             Primary    Alert    CSSA    Host               |
|                                                                            |
|                 Return Code:  ____     ____     ____    ____               |
|                 Error Class:  ____     ____     ____    ____               |
|                 Error Type :  ____     ____     ____    ____               |
+----------------------------------------------------------------------------+

F1=Help  F3=End  F10=API
```

Figure 112. Alert Interface Panel

The Alert Interface panel is used to exercise the functions provided by the alert API. These functions are:

Function	**Description**
Open	Open the alert API. This function must be executed before any other function.
Send	Send an alert NMVT through the alert API to a selected destination.
Close	Close the alert API.

Function Keys

There are no additional active function keys on this panel, aside from those described in "Function Keys" on page 174.

Parameters

The parameters which must be entered on this panel are:

Request Code = **O|S|C**
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
S	Issues an ARB with a Send request to the API when F10 is pressed. This should result in an NMVT being sent to the selected destination.
C	Issues an ARB with a Close request to the API when F10 is pressed.

NMVT Filename = *filename.ext*
filename.ext is the DOS name of the file (in the current directory) containing the NMVT which is to be sent through the API.

Send Alert To = **L|H|B**
Determines the destination(s) to which an alert is sent.

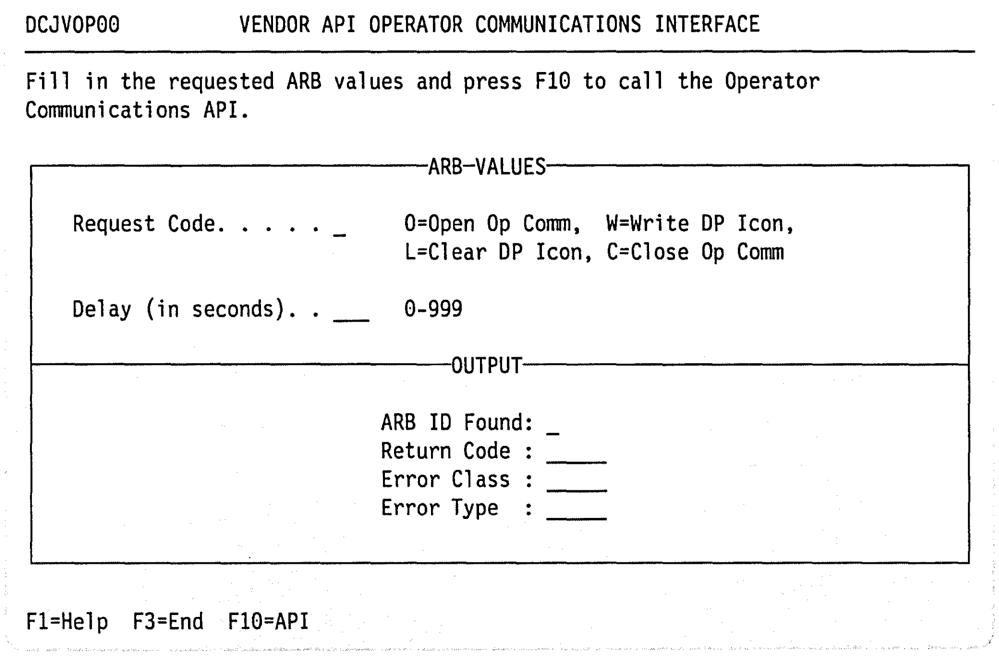
L (Local)	If Local is selected, the alert is sent only to the local alert facility.
H (Host)	If Host is selected, the alert is sent only to the host system to which NetView/PC is connected.
B (Both)	If Both is selected, the alert is sent to both the local and host systems.

Delay = *delay*
The sample program waits *delay* seconds before issuing the ARB to send the alert. This is a function of the sample program; *delay* is not a field in the ARB.

Output Fields

The column under **Primary** on the panel corresponds to the standard output fields described in "Other Information" on page 175 Additionally, the alert panel output includes displays of the alert (**Alert**), router (**CSSA**), and host (**Host**) error codes. Values and explanations of these return codes may be found in the main body of this document.

Operator Communications Interface



```
DCJVOP00         VENDOR API OPERATOR COMMUNICATIONS INTERFACE

Fill in the requested ARB values and press F10 to call the Operator
Communications API.

 ----------------------------ARB-VALUES------------------------------

   Request Code. . . . . _    O=Open Op Comm,  W=Write DP Icon,
                              L=Clear DP Icon, C=Close Op Comm

   Delay (in seconds). . ___  0-999
 ------------------------------OUTPUT--------------------------------

                            ARB ID Found: _
                            Return Code : ____
                            Error Class : ____
                            Error Type  : ____

F1=Help  F3=End  F10=API
```

Figure 113. Operator Communications Interface Panel

The Operator Communications panel is used to exercise the functions provided by the operator communications API. These functions are:

Function	**Description**
Open	Open the API. This function must be executed before any other function.
Write	Write a **DP** icon to the screen.
Clear	Clear the **DP** icon from the screen.
Close	Close the API.

Function Keys

There are no additional active function keys on this panel, aside from those described in "Function Keys" on page 174.

Parameters

The parameters which must be entered on this panel are:

Request Code = O|W|L|C
Determines the type of request to place in the ARB.

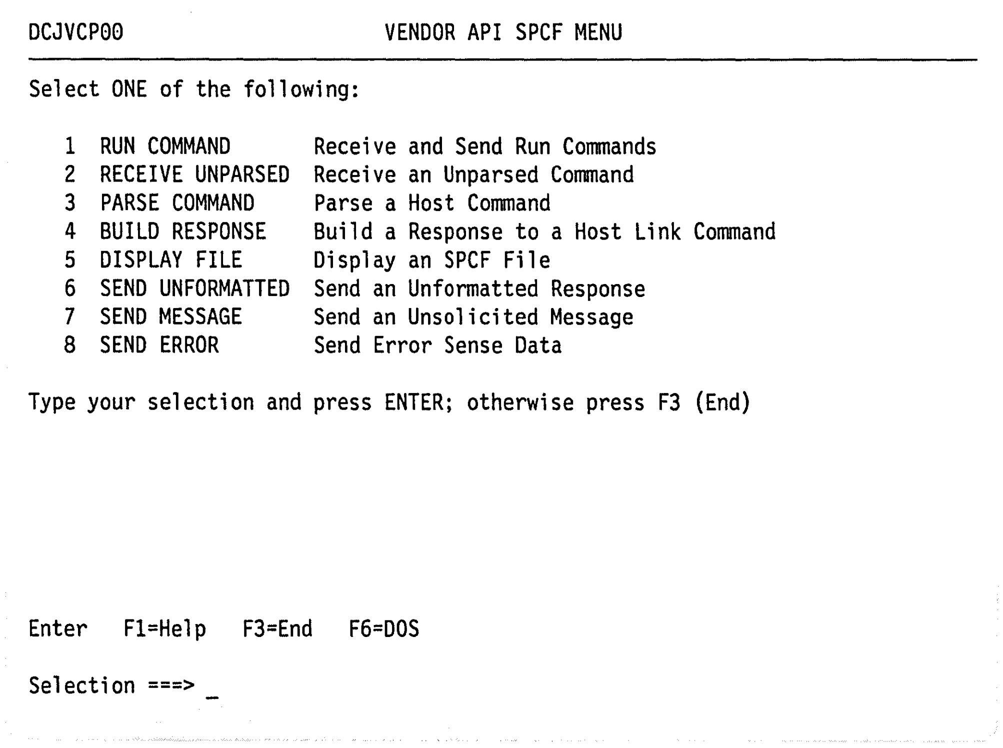
O	Issues an ARB with an Open request to the API when F10 is pressed.
W	Issues an ARB with a Write request to the API when F10 is pressed. This should result in the **DP** icon being displayed on the screen.
L	Issues an ARB with a Clear request to the API when F10 is pressed. This should result in the **DP** icon being cleared from the screen.
C	Issues an ARB with a Close request to the API when F10 is pressed.

Delay = *delay*
The amount of time to wait before displaying the **DP** icon is determined by *delay*. This is actually the amount of time the API Sample Program waits before issuing the ARB; it is not a field in the ARB itself.

Output Fields

There are no output fields on this panel, aside from the fields described in "Other Information" on page 175.

Service Point Command Facility Interface



```
DCJVCP00                      VENDOR API SPCF MENU
________________________________________________________________________________

Select ONE of the following:

   1  RUN COMMAND        Receive and Send Run Commands
   2  RECEIVE UNPARSED   Receive an Unparsed Command
   3  PARSE COMMAND      Parse a Host Command
   4  BUILD RESPONSE     Build a Response to a Host Link Command
   5  DISPLAY FILE       Display an SPCF File
   6  SEND UNFORMATTED   Send an Unformatted Response
   7  SEND MESSAGE       Send an Unsolicited Message
   8  SEND ERROR         Send Error Sense Data

Type your selection and press ENTER; otherwise press F3 (End)




Enter   F1=Help   F3=End   F6=DOS

Selection ===> _
```

Figure 114. SPCF Main Menu

The Service Point Command Facility Main Menu is used to access the SPCF services in the NetView/PC API Sample Program.

Function Keys

There are no additional active function keys, aside from those described in "Function Keys" on page 174.

Parameters

The only entry parameter on this panel is:

Selection = *n*

n is a number from one to eight corresponding to the number of the desired function on the SPCF menu (Figure 114). The functions presently available are:

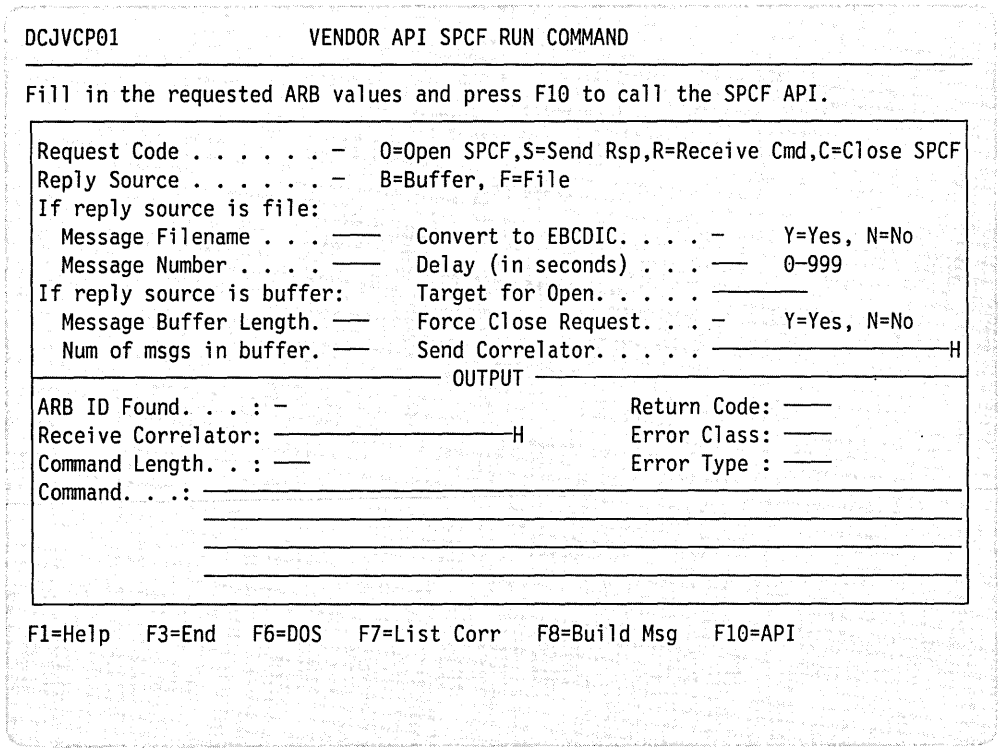
RUN	Receive and send **Run** commands ("SPCF Run Command" on page 182).
RECEIVE	Receive an unparsed command ("SPCF Receive Unparsed Command" on page 185).
PARSE	Parse a host command ("SPCF Parse Host Command" on page 187).

BUILD	Build a response to host **Link** commands ("SPCF Build Response" on page 189).
DISPLAY	Display an SPCF file ("SPCF Display File" on page 190).
RESPONSE	Send an unformatted response to the host ("SPCF Send Unformatted Response" on page 200).
MESSAGE	Send an unsolicited message to the host ("SPCF Send Unsolicited Message" on page 202).
ERROR	Send error sense data to the host ("SPCF Send Error Sense Data" on page 204).

Output Fields

There are no output fields on this panel.

SPCF Run Command



```
DCJVCP01                VENDOR API SPCF RUN COMMAND

Fill in the requested ARB values and press F10 to call the SPCF API.

Request Code . . . . . . _   O=Open SPCF,S=Send Rsp,R=Receive Cmd,C=Close SPCF
Reply Source . . . . . . _   B=Buffer, F=File
If reply source is file:
  Message Filename . . . ___    Convert to EBCDIC. . . . _      Y=Yes, N=No
  Message Number . . . . ___    Delay (in seconds) . . . ___    0-999
If reply source is buffer:      Target for Open. . . . . ________
  Message Buffer Length. ___    Force Close Request. . . _      Y=Yes, N=No
  Num of msgs in buffer. ___    Send Correlator. . . . . ___________________H
                                 OUTPUT
ARB ID Found. . . : _                              Return Code: ___
Receive Correlator: ____________________H          Error Class: ___
Command Length. . : ___                            Error Type : ___
Command. . .: _______________________________________________________________
              _______________________________________________________________
              _______________________________________________________________
              _______________________________________________________________

F1=Help   F3=End   F6=DOS   F7=List Corr   F8=Build Msg   F10=API
```

Figure 115. SPCF Run Command Panel

The Run Command panel is used to receive **Run** commands from the host or to send replies to previously received commands. The functions available from this panel are:

Function	Description
Open	Open the API. This function must be executed before any other function.
Send	Send a reply to a **Run** command.
Receive	Receive a **Run** command.
Close	Close the API.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active:

Key	Description
F7	Lists the correlators received, and allows selection of one of these active correlators (refer to "SPCF Correlator List" on page 208 for more information).
F8	Brings up a panel to allow the user to build a reply message. This panel is described in "SPCF Build Message" on page 207.

Parameters

The parameters which must be entered on this panel are:

Request Code = **O|S|R|C**
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
S	Issues an ARB with a Send request to the API when F10 is pressed. This should result in reply being sent to the host, with the reply being taken from the source selected by **Reply Source**.
R	Issues an ARB with a Receive request to the API when F10 is pressed. This function receives a **Run** command from the host.
C	Issues an ARB with a Close request to the API when F10 is pressed.

Reply Source = **B|F**
Determines the source from which a reply will be obtained.

B	If **B** is selected, the reply will be sent from the buffer. (Key F8 should be pressed in order to build the message). Fields **Message Buffer Length** and **Message Buffer Count** must also be set if this option is selected.
F	Indicates that the reply will be sent from a file. If this option is selected, fields **Message Filename** and **Message Number** must be entered.

Message Filename = *filename*
filename is the name of the file from which replies are to be read.

Note: The message filename follows EZVU naming conventions, i.e. it is of the format *cccc*.**MSG**, where *cccc* is a four character name; as an example, if the message file was **VAPI.MSG**, *filename* would be set to **VAPI**. This field is required if the **Reply Source** is set to **F**.

Message Number = *nnnn*
The message in the file numbered *nnnn* is the one that will be sent to the host. This field is required if the **Reply Source** is set to **F**.

Message Buffer Length = *length*
length is the length of the message buffer. This field is required if the **Reply Source** is set to **B** and **Convert** is set to **N**.

Message Buffer Count = *count*
count specifies the number of messages to be sent from the buffer. This field is required if the **Reply Source** is set to **B**. If **Convert** is set to **N** then *count* must be set to **1**.

Convert = **Y|N**
Determines whether messages in a buffer are converted from ASCII to EBCDIC before being transmitted to the host.

Y	Indicates that ASCII to EBCDIC translation occurs.
N	Indicates that no translation occurs.

Note: The **Convert** field is applicable only when messages are being sent from a buffer (i.e. when **Reply Source** is set to **B**). Replies being sent from a file are **always** converted to EBCDIC.

Delay = *delay*
The amount of time to wait before issuing the ARB to the NetView/PC API is determined by *delay*.

Target For Open = *name*
name specifies to the SPCF Router the name by which this application will be known to the NetView host.

Force Close Request = **Y|N**
This field is meaningful only on a close request (i.e. when **Request Code** is set to **C**).

Y	Indicates that the API should close even if there are outstanding **Receive** or **Send** requests.
N	Indicates that the API should complete the close successfully only if no **Receive** requests were outstanding.

Send Correlator = *hexnum*
Each command received and each reply sent must have a correlator. *hexnum* is a 10-byte hexadecimal number specifying the correlator which is to be sent to the host. This Send Correlator must match an already received Receive Correlator. A list of active correlators may be displayed by pressing F7. Refer

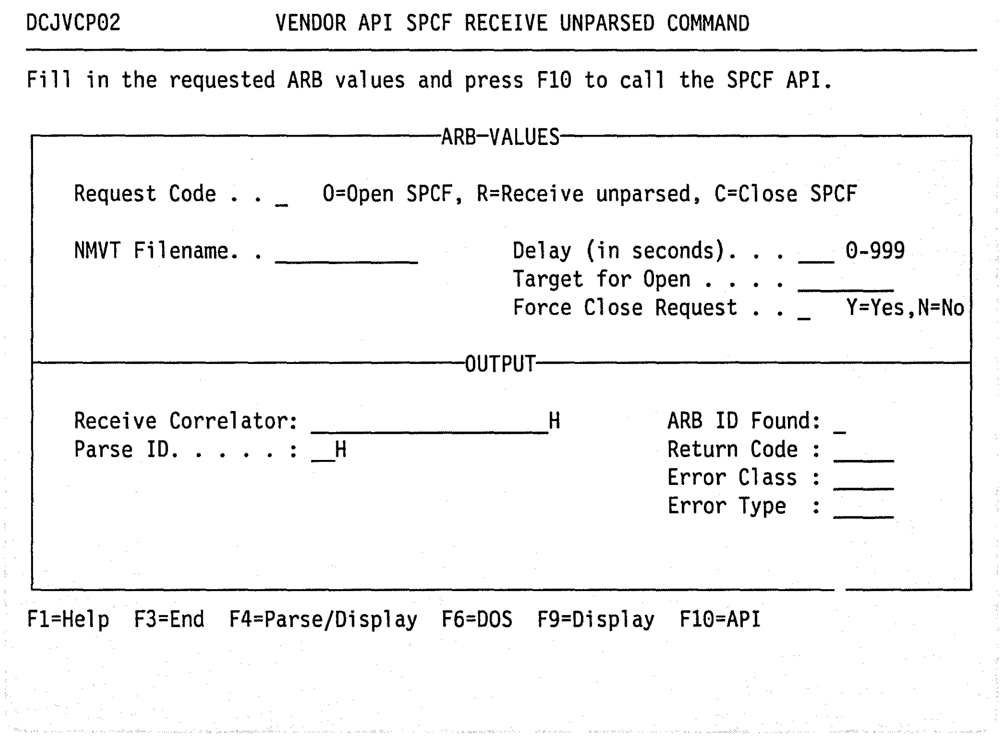
to "SPCF Correlator List" on page 208 for information regarding the correlator display panel.

Output Fields

In addition to the standard output fields described in "Other Information" on page 175, the following information is also provided on this panel:

Receive Correlator	The correlator is returned when a command is received from the Host. A reply to this command requires that the Send correlator in the ARB Values section match this correlator.
Command Length	Set to the length of the command received from the host. The SPCF Router supplies this length on a GET RUN request.
Command	The **command** field contains the actual text of the command received.

SPCF Receive Unparsed Command



```
DCJVCP02              VENDOR API SPCF RECEIVE UNPARSED COMMAND

Fill in the requested ARB values and press F10 to call the SPCF API.

                          ARB-VALUES

   Request Code . . _   O=Open SPCF, R=Receive unparsed, C=Close SPCF

   NMVT Filename. . ____________      Delay (in seconds). . . ___ 0-999
                                      Target for Open . . . . ________
                                      Force Close Request . . _   Y=Yes,N=No

                            OUTPUT

   Receive Correlator: ____________________H       ARB ID Found: _
   Parse ID. . . . . : __H                         Return Code : _____
                                                   Error Class : _____
                                                   Error Type  : _____

F1=Help  F3=End  F4=Parse/Display  F6=DOS  F9=Display  F10=API
```

Figure 116. SPCF Get No Parse Panel

The Receive Unparsed command panel is used to receive SPCF NMVTs from the host. The functions available from this panel are:

Function	**Description**
Open	Open the API. This function must be executed before any other function.
Receive Unparsed	Receive an unparsed command from the host. (In order to parse the received NMVT, see "SPCF Parse Host Command" on page 187).
Close	Close the API.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active:

Key	**Description**
F4	Calls the parse API to parse the received NMVT, and displays the results of the parse.

Parameters

The parameters which must be entered on this panel are:

Request Code = **O|R|C**
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
R	Issues an ARB with a Receive Unparsed request to the API when F10 is pressed.
C	Issues an ARB with a Close request to the API when F10 is pressed.

NMVT Filename = *filename.ext*
The SPCF NMVT received from the host will be placed in file *filename.ext* in the current directory. **Warning:** Any file with the same name will be overwritten, and its data lost.

Care should therefore be taken not to set *filename.ext* to an existing file; additionally, the F10 key should not be pressed more than once without changing *filename.ext*.

Delay = *delay*
The amount of time to wait before issuing the ARB to the NetView/PC API is determined by *delay*.

Target For Open = *name*
name specifies to the SPCF Router the name by which this application will be known to the NetView host.

Force Close Request = Y|N
This field is meaningful only on a close request (i.e. when **Request Code** is set to **C**).

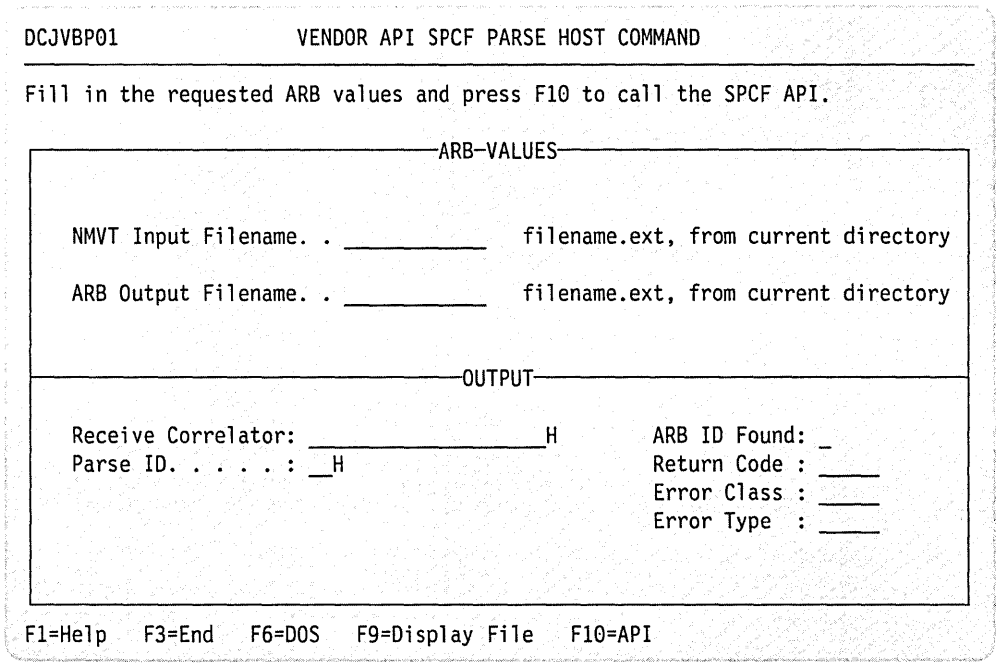
Y	Indicates that the API should close even if there are outstanding **Receive** requests.
N	Indicates that the API should complete the close successfully only if no **Receive** requests were outstanding.

Output Fields

In addition to the standard output fields described in "Other Information" on page 175, the following information is also provided on this panel:

Receive Correlator	The correlator is returned when a command is received from the Host. A reply to this command requires that the Send correlator in the ARB Values section match this correlator.
Parse ID	The actual PARSE ID as returned by the API subroutine. This hexadecimal number will match the last byte of the major vector key field in the received NMVT.

SPCF Parse Host Command



```
DCJVBP01                VENDOR API SPCF PARSE HOST COMMAND

Fill in the requested ARB values and press F10 to call the SPCF API.

                              ARB-VALUES

   NMVT Input Filename. . __________   filename.ext, from current directory

   ARB Output Filename. . __________   filename.ext, from current directory

                                OUTPUT

   Receive Correlator: ________________H        ARB ID Found: _
   Parse ID. . . . . : _H                       Return Code : ____
                                                Error Class : ____
                                                Error Type  : ____

F1=Help   F3=End   F6=DOS   F9=Display File   F10=API
```

Figure 117. SPCF Parse Command Panel

The SPCF Parse Host Command panel is used to parse an unformatted NMVT received from the host into a parse ARB.

Function Keys

There are no active function keys on this panel, aside from those described in "Function Keys" on page 174.

Parameters

The parameters which must be entered on this panel are:

NMVT Input Filename = *nmvtfile.ext*
The SPCF NMVT in *nmvtfile.ext* will be read into a buffer in the API Sample Program, and will be passed to the API as a parameter in the parse request ARB.

ARB Output Filename = *arbfname.ext*
If the NMVT in the selected file was parsed successfully, the resulting parse ARB is written to *arbfname.ext* in the current directory. **Warning:** Any file with the same name will be overwritten, and its data lost.

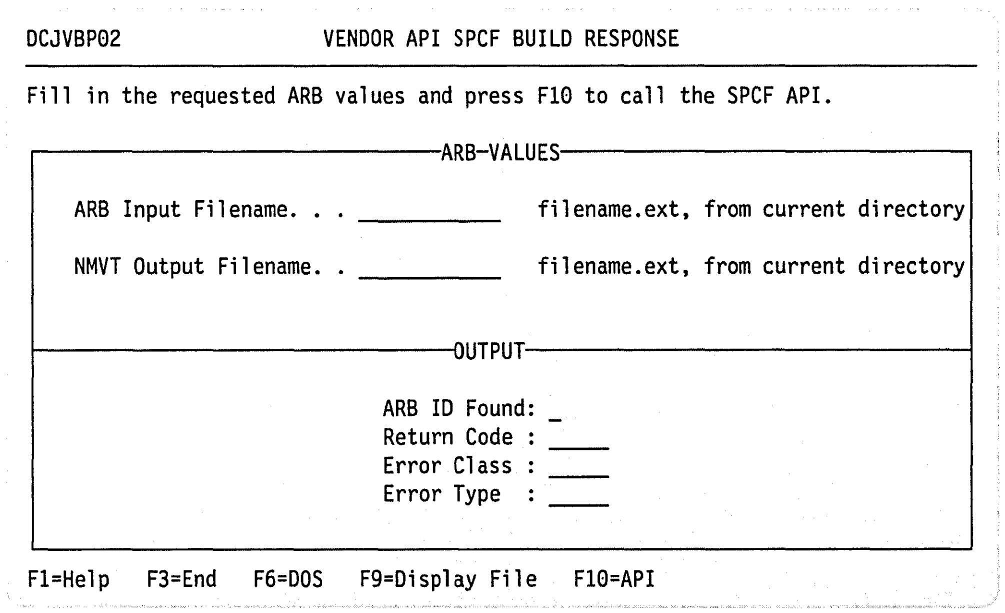
Note: Refer to "File Formats" on page 174 for information on the format of the NMVT and ARB files.

Output Fields

In addition to the standard output fields described in "Other Information" on page 175, the following information is also provided on this panel:

Receive Correlator	The correlator associated with the NMVT in the selected file.
Parse ID	The actual PARSE ID as returned by the API subroutine.

SPCF Build Response



```
DCJVBP02                     VENDOR API SPCF BUILD RESPONSE
Fill in the requested ARB values and press F10 to call the SPCF API.

                              ARB-VALUES
   ARB Input Filename. . . ___________   filename.ext, from current directory

   NMVT Output Filename. . ___________   filename.ext, from current directory

                                OUTPUT
                           ARB ID Found: _
                           Return Code : ____
                           Error Class : ____
                           Error Type  : ____

F1=Help   F3=End   F6=DOS   F9=Display File   F10=API
```

Figure 118. SPCF Build Response Panel

The SPCF Build Response Panel is used to build a response NMVT from data contained in a build ARB, which is read in from a disk file.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active:

Key	**Description**
F9	Displays a file. Refer to "SPCF Display File" on page 190 for more information.

Parameters

The parameters which must be entered on this panel are:

ARB Input Filename = *arbfname.ext*
The file *arbfname.ext* is assumed to contain a build ARB. The ARB is read into a buffer and placed in a format acceptable to the API.

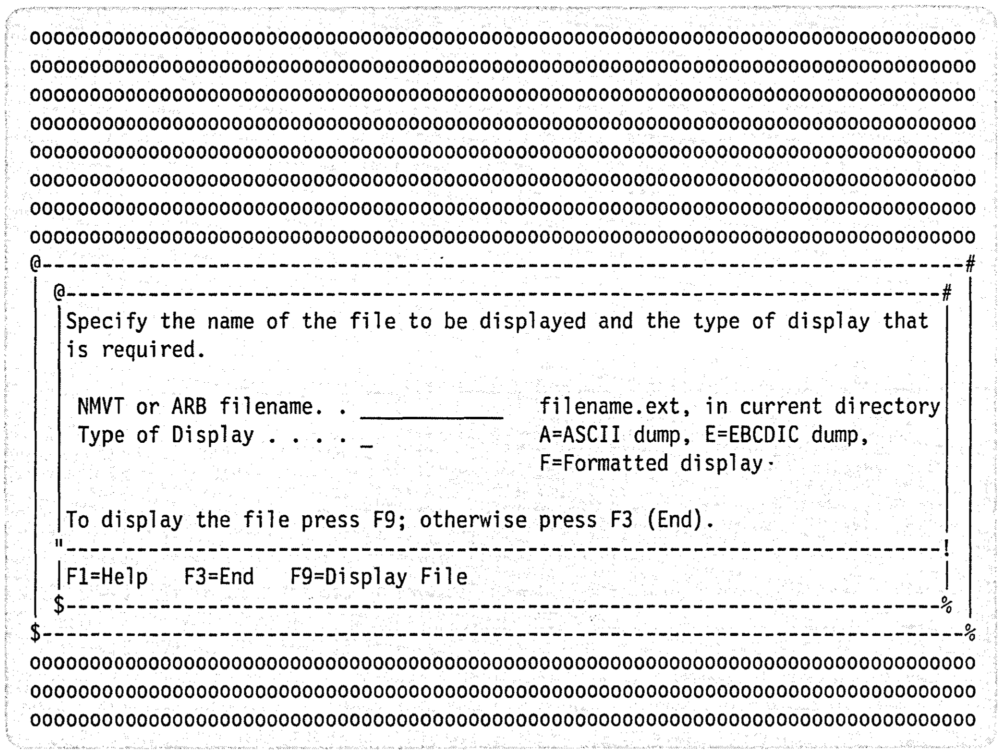
NMVT Output Filename = *nmvtfile.ext*
If the call to the build API is successful, the NMVT created will be placed in *nmvtfile.ext* in the current directory. **Warning:** If the file *nmvtfile.ext* already exists, it will be overwritten and any data it contains will be lost.

Note: Refer to "File Formats" on page 174 for information on the format of the NMVT and ARB files.

Output Fields

There are no output fields on this panel, aside from the general output information documented in "Other Information" on page 175.

SPCF Display File



```
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
@------------------------------------------------------------------------------#
| @------------------------------------------------------------------------# |
| |Specify the name of the file to be displayed and the type of display that | |
| |is required.                                                              | |
| |                                                                          | |
| | NMVT or ARB filename. . ____________   filename.ext, in current directory| |
| | Type of Display . . . . _              A=ASCII dump, E=EBCDIC dump,      | |
| |                                        F=Formatted display               | |
| |                                                                          | |
| |To display the file press F9; otherwise press F3 (End).                   | |
| "--------------------------------------------------------------------------! |
| |F1=Help   F3=End   F9=Display File                                        | |
| $--------------------------------------------------------------------------% |
$------------------------------------------------------------------------------%
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

Figure 119. SPCF Display File Pop-Up Panel

The API Sample Program includes a facility for displaying SPCF files. This facility may be invoked either from the SPCF main menu or from many of the other SPCF panels. When invoked, the pop-up panel shown in Figure 119 is displayed (the "o" characters in the figure represent the background panel).

The types of displays provided by this pop-up are described in:

Type	**Description**
Dump	(Unformatted) Displays hexadecimal and character dump; see "SPCF Unformatted Display Panel" on page 192.
Run	(Formatted) Displays the parse ARB when the parse ID indicates that the parsed NMVT contained a **Run** command; see "SPCF Display Run Command" on page 194.
Link PD	(Formatted) Displays the parse ARB when the parse ID indicates that the parsed NMVT contained a **Link PD** command; see "SPCF Display Link PD Command" on page 195.

Link Test	(Formatted) Displays the parse ARB when the parse ID indicates that the parsed NMVT contained a **Link Test** command; see "SPCF Display Link Test Command" on page 198.
Link Data	(Formatted) Displays the parse ARB when the parse ID indicates that the parsed NMVT contained a **Link Data** command; see "SPCF Display Link Data Command" on page 197.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active:

Key	Description
F9	Displays the selected file in the desired format.

Parameters

The parameters which must be entered on this panel are:

NMVT or ARB Filename = *filename.ext*
The file *filename.ext* is assumed to contain either an NMVT or a parse ARB written by the Parse Host Command portion of the API Sample Program.

Display Type = **A|E|F**
Determines the type of display desired.

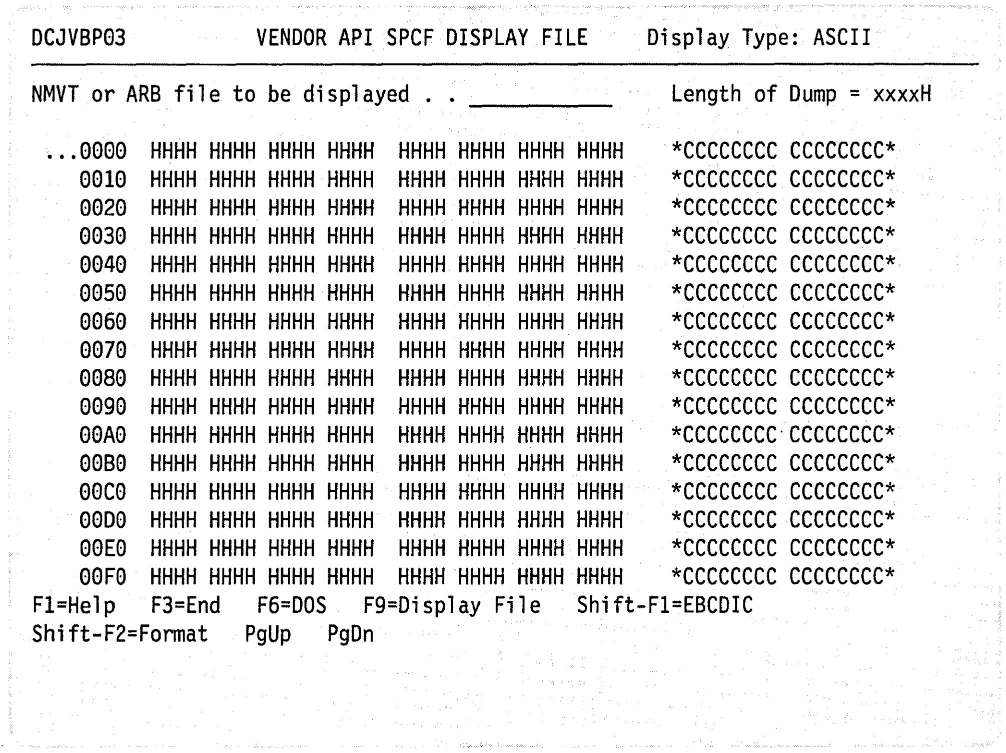
A	Indicates that the file is to be displayed in unformatted form, with text assumed to be encoded in ASCII.
E	Indicates that the file is to be displayed in unformatted form, with text assumed to be encoded in EBCDIC.
F	Indicates that a formatted display is to be provided.

Note: This selection is valid only if *filename.ext* contains a parse output ARB, as written by the Parse Host Command portion of the API Sample Program. If an attempt is made to view a file containing data which is not recognized as a parse ARB, an unformatted display will be provided instead, and an explanatory message placed on the screen.

Output Fields

There are no output fields in this panel.

SPCF Unformatted Display Panel



```
DCJVBP03          VENDOR API SPCF DISPLAY FILE     Display Type: ASCII
--------------------------------------------------------------------------
NMVT or ARB file to be displayed . . __________     Length of Dump = xxxxH

 ...0000  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0010  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0020  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0030  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0040  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0050  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0060  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0070  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0080  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    0090  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    00A0  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    00B0  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    00C0  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    00D0  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    00E0  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
    00F0  HHHH HHHH HHHH HHHH  HHHH HHHH HHHH HHHH    *CCCCCCCC CCCCCCCC*
F1=Help   F3=End   F6=DOS   F9=Display File   Shift-F1=EBCDIC
Shift-F2=Format   PgUp   PgDn
```

Figure 120. SPCF Display Unformatted Format Panel

This panel provides a dump of a selected SPCF NMVT or ARB file.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active:

Key	Description
F9	Causes the specified file to be read into memory and displayed beginning at the specified offset.
Shift-F11	Switches the character translation between ASCII and EBCDIC.
Shift-F12	Attempts to provide a formatted display of the data in the buffer.

Note: Changing the filename on the panel and pressing Shift-F12 will **not** cause a display of the new file. Only the F9 key will cause new data to be read in for display.

PgUp	Displays data one page closer to the start of the buffer in which the file has been placed, i.e. the offset into the buffer is **decreased** by 240 decimal (F0 hex) bytes.
PgDn	Displays data one page closer to the end of the buffer in which the file has been placed, i.e. the offset into the buffer is **increased** by 240 decimal (F0 hex) bytes.

Parameters

The input fields on this panel are:

NMVT or ARB Filename to be displayed = *filename.ext*
The name of a file in the current directory which is to be displayed.

{Offset} = *xxxx*
If the first offset field (see description of output fields below) is overtyped, the next time F9 is pressed the offset into the selected file at which the dump display begins will be set to the hexadecimal number *xxxx*.

Note: The F9 key **must** be pressed in order for changes to either the filename or the offset to be processed. When *filename.ext* or *xxxx* are changed and a function key other than F9 is pressed, the changes are undone before any processing takes place.

Output Fields

The items displayed by this panel are:

Field	Description
Offset	This is the first column in the display. Each row in this column contains the offset into the buffer (and thus the file) of the first byte of the data in the columns to the right.
Hex Dump	Hexadecimal representation of the data in the selected file. (These are the columns marked with an **H**).
Character Dump	Character representation of the data in the selected file. These are the columns in the display marked with a **C**. The data may be interpreted as either ASCII or EBCDIC; this may be switched back and forth using the Shift-F11 key. Non-displayable characters, as well as the PC box graphics characters, are represented by a period (".").
Length of Dump	Set to the length of the data in the selected file.

SPCF Display Run Command

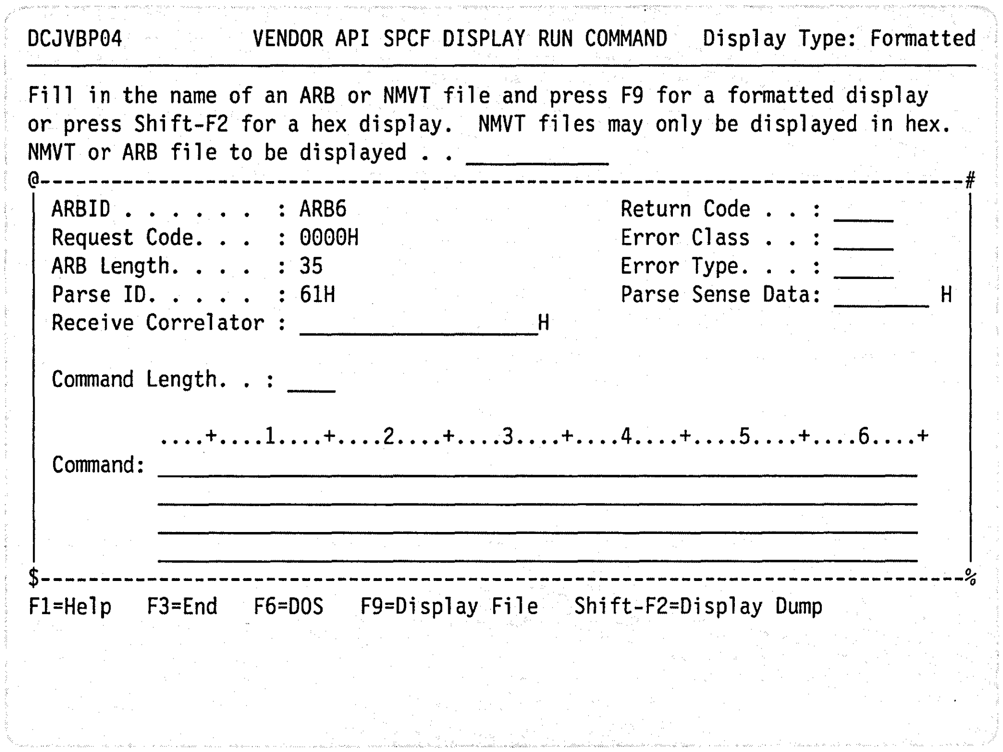

```
DCJVBP04          VENDOR API SPCF DISPLAY RUN COMMAND   Display Type: Formatted
-------------------------------------------------------------------------------
Fill in the name of an ARB or NMVT file and press F9 for a formatted display
or press Shift-F2 for a hex display.  NMVT files may only be displayed in hex.
NMVT or ARB file to be displayed . . ________
@----------------------------------------------------------------------------#
| ARBID . . . . . . : ARB6                      Return Code . . : ___        |
| Request Code. . .  : 0000H                    Error Class . . : ___        |
| ARB Length. . . .  : 35                       Error Type. . . : ___        |
| Parse ID. . . . .  : 61H                      Parse Sense Data: ______ H   |
| Receive Correlator : _______________H                                      |
|                                                                            |
| Command Length. . : ___                                                    |
|                                                                            |
|          ....+....1....+....2....+....3....+....4....+....5....+....6....+ |
| Command: ____________________________________________________________      |
|          ____________________________________________________________      |
|          ____________________________________________________________      |
|          ____________________________________________________________      |
$----------------------------------------------------------------------------%
F1=Help   F3=End   F6=DOS   F9=Display File   Shift-F2=Display Dump
```

Figure 121. SPCF Display Formatted Run Command Panel

This is one of the four types of formatted display panels. It is displayed when a formatted display is requested of a parse ARB used to parse a **Run** command NMVT.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active on this panels:

Key	**Description**
F9	Causes the specified file to be read into memory and an attempt made to provide a formatted display of the data in the file.
Shift-F12	Provides an unformatted display of the data in the buffer. **Note:** Changing the filename on the panel and pressing Shift-F12 will **not** cause a display of the new file. Only the F9 key will cause new data to be read in for display.

Parameters

The input fields on this panel are:

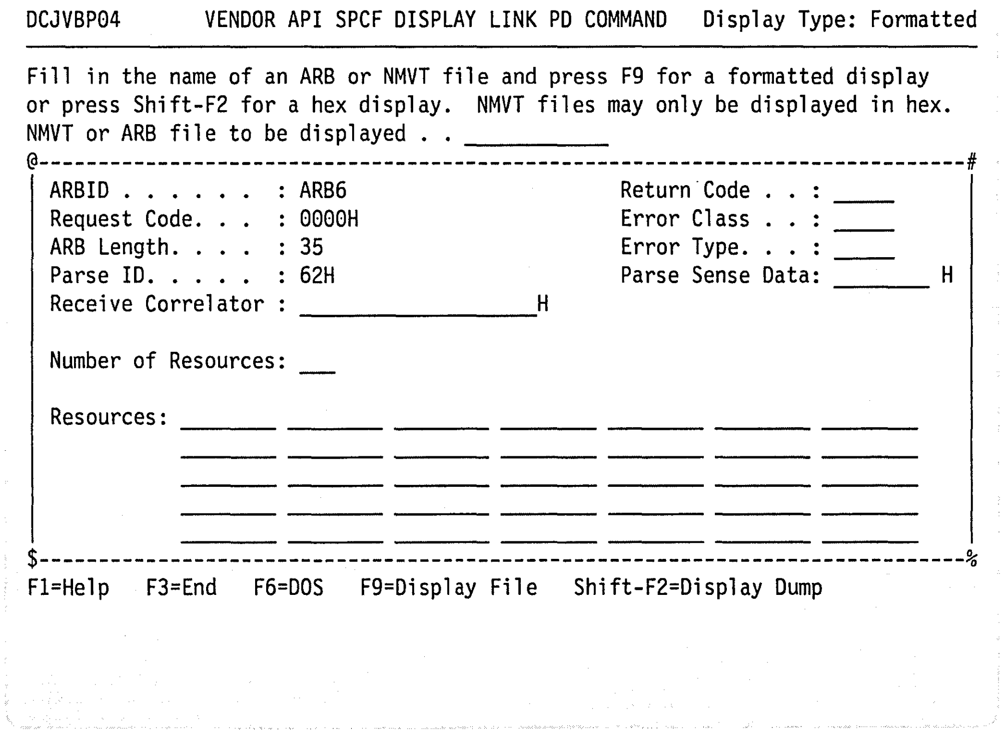
NMVT or ARB Filename to be displayed = *filename.ext*
The name of a file in the current directory which is to be displayed.

Output Fields

The following information is displayed on the panel:

Field	Description
Receive Correlator	A 20-character hexadecimal representation of the 10-byte correlator found in the selected file.
Parse Sense Data	An eight-character hexadecimal representation of the four-byte field parse sense field in the parse ARB.
Command Length	The decimal length of the parsed **Run** command is returned in this field.
Command	The actual **Run** command is displayed here.

SPCF Display Link PD Command



```
DCJVBP04       VENDOR API SPCF DISPLAY LINK PD COMMAND   Display Type: Formatted

Fill in the name of an ARB or NMVT file and press F9 for a formatted display
or press Shift-F2 for a hex display.  NMVT files may only be displayed in hex.
NMVT or ARB file to be displayed . . ____________
@---------------------------------------------------------------------------#
| ARBID . . . . . . : ARB6                        Return Code . . : _____   |
| Request Code. . . : 0000H                       Error Class . . : _____   |
| ARB Length. . . . : 35                          Error Type. . . : _____   |
| Parse ID. . . . . : 62H                         Parse Sense Data: ________ H|
| Receive Correlator : ____________________H                                |
|                                                                           |
| Number of Resources: ___                                                  |
|                                                                           |
| Resources: ________ ________ ________ ________ ________ ________ ________ |
|            ________ ________ ________ ________ ________ ________ ________ |
|            ________ ________ ________ ________ ________ ________ ________ |
|            ________ ________ ________ ________ ________ ________ ________ |
|            ________ ________ ________ ________ ________ ________ ________ |
$---------------------------------------------------------------------------%
F1=Help   F3=End   F6=DOS   F9=Display File   Shift-F2=Display Dump
```

Figure 122. SPCF Display Formatted Link PD Command Panel

This is one of the four types of formatted display panels. It is displayed when a formatted display is requested of a parse ARB which has been used to parse a **Link PD** NMVT.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active on this panels:

Key	Description
F9	Causes the specified file to be read into memory and an attempt made to provide a formatted display of the data in the file.
Shift-F12	Provides an unformatted display of the data in the buffer. **Note:** Changing the filename on the panel and pressing Shift-F12 will **not** cause a display of the new file. Only the F9 key will cause new data to be read in for display.

Parameters

The input fields on this panel are:

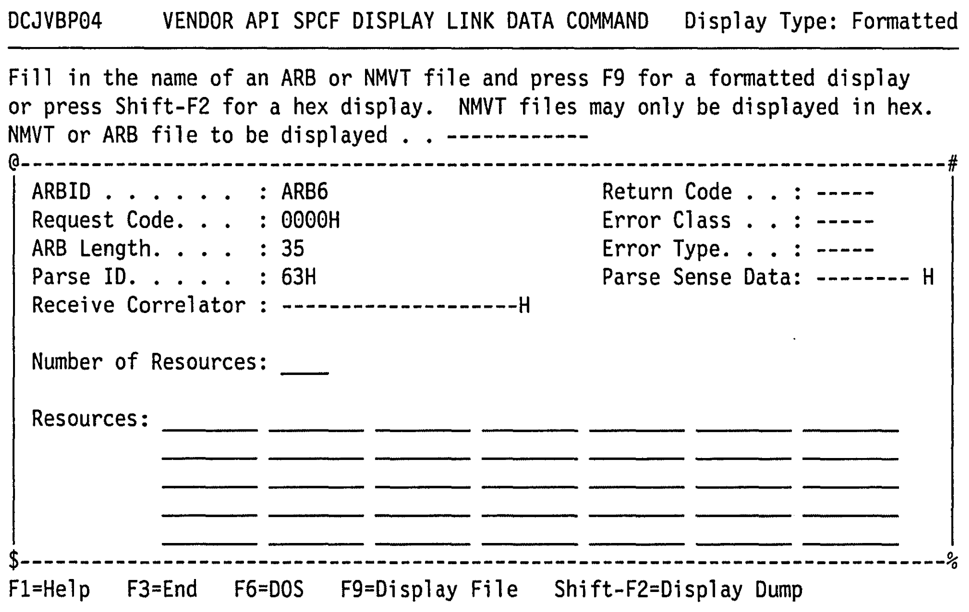
NMVT or ARB Filename to be displayed = *filename.ext*
The name of a file in the current directory which is to be displayed.

Output Fields

The following information is displayed on the panel:

Field	Description
Receive Correlator	A 20-character hexadecimal representation of the 10-byte correlator found in the selected file.
Parse Sense Data	An eight-character hexadecimal representation of the four-byte field parse sense field in the parse ARB.
Number of Resources	The number of resource names in the parsed NMVT.
Resources	A list of the resource names that were present in the parsed NMVT.

SPCF Display Link Data Command



```
DCJVBP04     VENDOR API SPCF DISPLAY LINK DATA COMMAND   Display Type: Formatted
--------------------------------------------------------------------------------
Fill in the name of an ARB or NMVT file and press F9 for a formatted display
or press Shift-F2 for a hex display.  NMVT files may only be displayed in hex.
NMVT or ARB file to be displayed . . ------------
@------------------------------------------------------------------------------#
| ARBID . . . . . . : ARB6                        Return Code . . : -----      |
| Request Code. . .  : 0000H                      Error Class . . : -----      |
| ARB Length. . . .  : 35                         Error Type. . . : -----      |
| Parse ID. . . . .  : 63H                        Parse Sense Data: -------- H |
| Receive Correlator : --------------------H                                   |
|                                                                              |
| Number of Resources: ____                                                    |
|                                                                              |
| Resources: _______ _______ _______ _______ _______ _______ _______          |
|            _______ _______ _______ _______ _______ _______ _______          |
|            _______ _______ _______ _______ _______ _______ _______          |
|            _______ _______ _______ _______ _______ _______ _______          |
|            _______ _______ _______ _______ _______ _______ _______          |
$------------------------------------------------------------------------------%
F1=Help    F3=End    F6=DOS    F9=Display File    Shift-F2=Display Dump
```

Figure 123. SPCF Display Formatted Link Data Command Panel

This is one of the four types of formatted display panels. It is displayed when a formatted display is requested of a parse ARB which has been used to parse a **Link Data** NMVT.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active on this panels:

Key	Description
F9	Causes the specified file to be read into memory and an attempt made to provide a formatted display of the data in the file.
Shift-F12	Provides an unformatted display of the data in the buffer.

Note: Changing the filename on the panel and pressing Shift-F12 will **not** cause a display of the new file. Only the F9 key will cause new data to be read in for display.

Parameters

The input fields on this panel are:

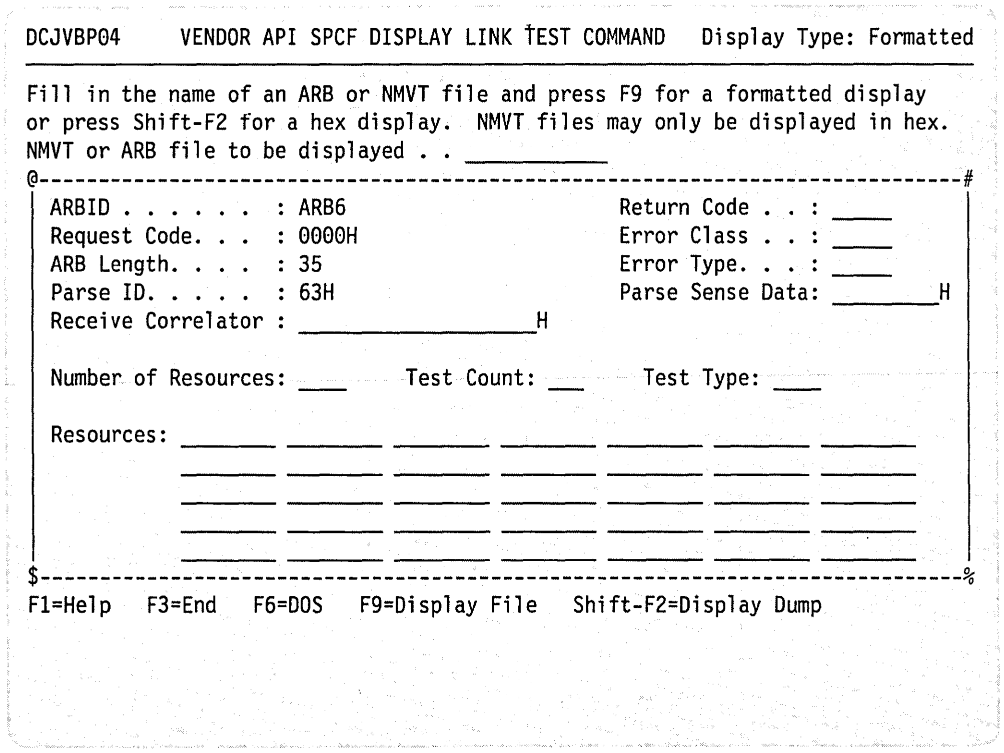
NMVT or ARB Filename to be displayed = *filename.ext*
The name of a file in the current directory which is to be displayed.

Output Fields

The following information is displayed on the panel:

Field	Description
Receive Correlator	A 20-character hexadecimal representation of the 10-byte correlator found in the selected file.
Parse Sense Data	An eight-character hexadecimal representation of the four-byte field parse sense field in the parse ARB.
Number of Resources	The number of resource names in the parsed NMVT.
Resources	A list of the resource names that were present in the parsed NMVT.

SPCF Display Link Test Command



```
DCJVBP04     VENDOR API SPCF DISPLAY LINK TEST COMMAND   Display Type: Formatted
--------------------------------------------------------------------------------
Fill in the name of an ARB or NMVT file and press F9 for a formatted display
or press Shift-F2 for a hex display.  NMVT files may only be displayed in hex.
NMVT or ARB file to be displayed . . ____________
@------------------------------------------------------------------------------#
| ARBID . . . . . . : ARB6                          Return Code . . : _____    |
| Request Code. . . : 0000H                         Error Class . . : _____    |
| ARB Length. . . . : 35                            Error Type. . . : _____    |
| Parse ID. . . . . : 63H                           Parse Sense Data: _________H |
| Receive Correlator : ____________________H                                    |
|                                                                               |
| Number of Resources: ____    Test Count: ___    Test Type: ____              |
|                                                                               |
| Resources: ________ ________ ________ ________ ________ ________ ________    |
|            ________ ________ ________ ________ ________ ________ ________    |
|            ________ ________ ________ ________ ________ ________ ________    |
|            ________ ________ ________ ________ ________ ________ ________    |
|            ________ ________ ________ ________ ________ ________ ________    |
$------------------------------------------------------------------------------%
F1=Help   F3=End   F6=DOS   F9=Display File   Shift-F2=Display Dump
```

Figure 124. SPCF Display Formatted Link Test Command Panel

This is one of the four types of formatted display panels. It is displayed when a formatted display is requested of a parse ARB which has been used to parse a **Link Test** NMVT.

Function Keys

In addition to the function keys described in "Function Keys" on page 174, the following keys are also active on this panels:

Key	Description
F9	Causes the specified file to be read into memory and an attempt made to provide a formatted display of the data in the file.
Shift-F12	Provides an unformatted display of the data in the buffer. **Note:** Changing the filename on the panel and pressing Shift-F12 will **not** cause a display of the new file. Only the F9 key will cause new data to be read in for display.

Parameters

The input fields on this panel are:

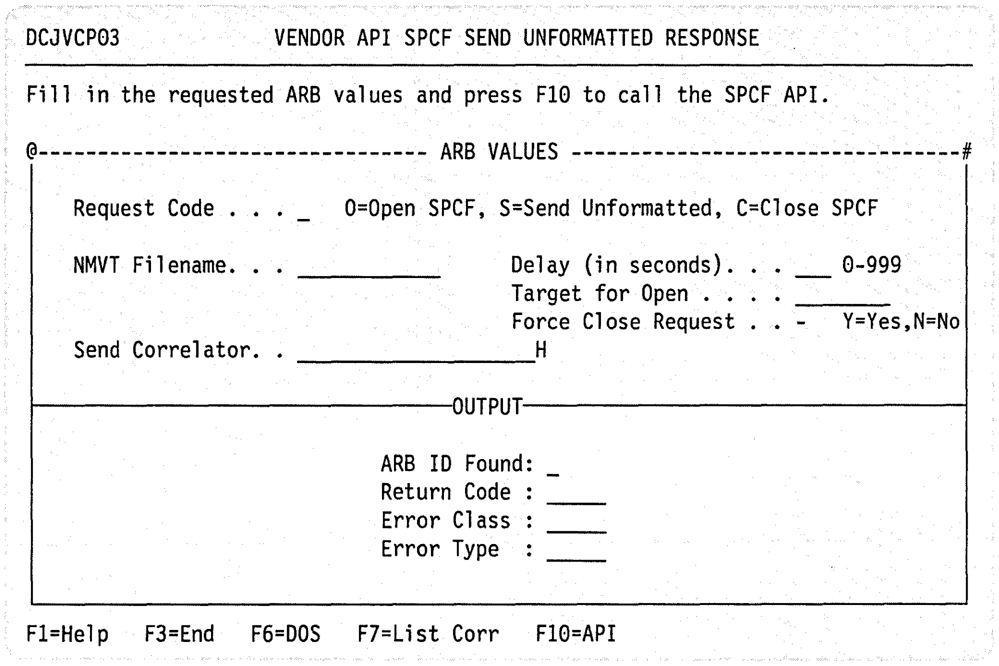
NMVT or ARB Filename to be displayed = *filename.ext*
The name of a file in the current directory which is to be displayed.

Output Fields

The following information is displayed on the panel:

Field	Description
Receive Correlator	A 20-character hexadecimal representation of the 10-byte correlator found in the selected file.
Parse Sense Data	An eight-character hexadecimal representation of the four-byte field parse sense field in the parse ARB.
Number of Resources	The number of resource names in the parsed NMVT.
Test Count	The test count specified in the parsed NMVT.
Test Type	The type of test specified in the parsed NMVT.
Resources	A list of the resource names that were present in the parsed NMVT.

SPCF Send Unformatted Response



```
DCJVCP03              VENDOR API SPCF SEND UNFORMATTED RESPONSE

Fill in the requested ARB values and press F10 to call the SPCF API.

@-------------------------------- ARB VALUES --------------------------------#

    Request Code . . . _   O=Open SPCF, S=Send Unformatted, C=Close SPCF

    NMVT Filename. . . ____________      Delay (in seconds). . . ___ 0-999
                                         Target for Open . . . . ________
                                         Force Close Request . . -   Y=Yes,N=No
    Send Correlator. . ____________________H

-------------------------------------OUTPUT-----------------------------------

                                 ARB ID Found: _
                                 Return Code : _____
                                 Error Class : _____
                                 Error Type  : _____

F1=Help   F3=End   F6=DOS   F7=List Corr   F10=API
```

Figure 125. SPCF Put Unformatted Panel

The Send Unformatted Response panel is used to put unformatted responses to the host. The functions available from this panel are:

Function	**Description**
Open	Open the API. This function must be executed before any other function.
Send	Send a response NMVT to the host.
Close	Close the API.

Function Keys

The functions keys that are active on this panel, aside from the ones described in "Function Keys" on page 174, are:

Key	**Description**
F7	Lists the correlators received, and allows selection of one of these active correlators (refer to "SPCF Correlator List" on page 208 for more information).

Parameters

The parameters which must be entered on this panel are:

Request Code = **O|S|C**
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
S	Issues an ARB with a Send request to the API when F10 is pressed. This should result in the NMVT being sent to the host.
C	Issues an ARB with a Close request to the API when F10 is pressed.

NMVT Filename = *nmvtfile.ext*
The SPCF NMVT found in *nmvtfile.ext* in the current directory will be sent to the host.

Send Correlator = *hexnum*
Each command received and each reply sent must have a correlator. *hexnum* is a 10-byte hexadecimal number specifying the correlator which is to be sent to the host. This Send Correlator must match an already received Receive Correlator. A list of active correlators may be displayed by pressing F7. Refer to "SPCF Correlator List" on page 208 for information regarding the correlator display panel.

Delay = *delay*
The amount of time to wait before issuing the ARB to the NetView/PC API is determined by *delay*.

Target For Open = *name*
name specifies to the SPCF Router the name by which this application will be known to the NetView host.

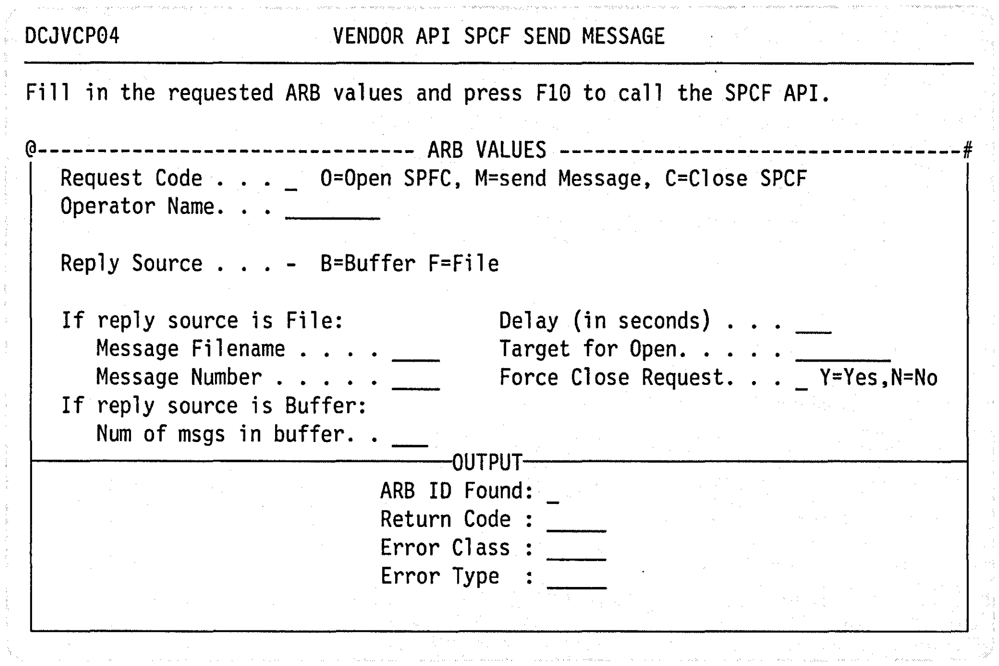
Force Close Request = **Y|N**
This field is meaningful only on a close request (i.e. when **Request Code** is set to **C**).

Y	Indicates that the API should close even if there are outstanding **Receive** requests.
N	Indicates that the API should complete the close successfully only if no **Receive** requests were outstanding.

Output Fields

There are no output fields on this panel, aside from the general output information documented in "Other Information" on page 175.

SPCF Send Unsolicited Message



```
DCJVCP04                  VENDOR API SPCF SEND MESSAGE
________________________________________________________________________
Fill in the requested ARB values and press F10 to call the SPCF API.

@-------------------------------- ARB VALUES ---------------------------------#
|  Request Code . . . _  O=Open SPFC, M=send Message, C=Close SPCF
|  Operator Name. . . ________
|
|  Reply Source . . . -  B=Buffer F=File
|
|  If reply source is File:           Delay (in seconds) . . . ___
|     Message Filename . . . . ____   Target for Open. . . . . ________
|     Message Number . . . . . ____   Force Close Request. . . _ Y=Yes,N=No
|  If reply source is Buffer:
|     Num of msgs in buffer. . ___
|--------------------------------OUTPUT-----------------------------------------
|                                ARB ID Found: _
|                                Return Code : _____
|                                Error Class : _____
|                                Error Type  : _____
|_______________________________________________________________________________
```

Figure 126. SPCF Send Message Panel

This panel is used to send an unsolicited message to a NetView host. The functions available from this panel are:

Function	Description
Open	Open the API. This function must be executed before any other function.
Send	Send a message.
Close	Close the API.

Function Keys

The functions keys that are active on this panel, aside from the ones described in "Function Keys" on page 174, are:

Key	Description
F8	Brings up a panel to allow the user to build a message. This panel is described in "SPCF Build Message" on page 207.

Parameters

The parameters which must be entered on this panel are:

Request Code = **O|M|C**
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
M	Issues an ARB with a send request to the API when F10 is pressed. This should result in a Message being sent to the host.
C	Issues an ARB with a Close request to the API when F10 is pressed.

Operator Name = *opername*
opername is the one to eight character operator name matching the Netview session.

Reply Source = **B|F**
Determines the source from which a reply will be obtained.

B	If **B** is selected, the reply will be sent from the buffer. Field **Message Buffer Count** must also be set if this option is selected.
F	Indicates that the reply will be sent from a file. If this option is selected, fields **Message Filename** and **Message Number** must be entered.

Message Filename = *msgf*
msgf is the four character name of the file (in the current directory) from which replies are to be read. This file must be in the same format as EZVU message files, and an extension of **.MSG** is assumed. This field is required if the **Reply Source** is set to **F**.

Message Number = *nnnn*
Message number *nnnn* in the file specified by **Message Filename** is the one that will be sent to the host. This field is required if the **Reply Source** is set to **F**.

Num of Msgs in Buffer = *count*
count specifies the number of messages to be sent from the buffer. This field is required if the **Reply Source** is set to **B**.

Delay = *delay*
The amount of time to wait before issuing the ARB to the NetView/PC API is determined by *delay*.

Target For Open = *name*
name specifies to the SPCF Router the name by which this application will be known to the NetView host.

Force Close Request = **Y|N**
This field is meaningful only on a close request (i.e. when **Request Code** is set to **C**).

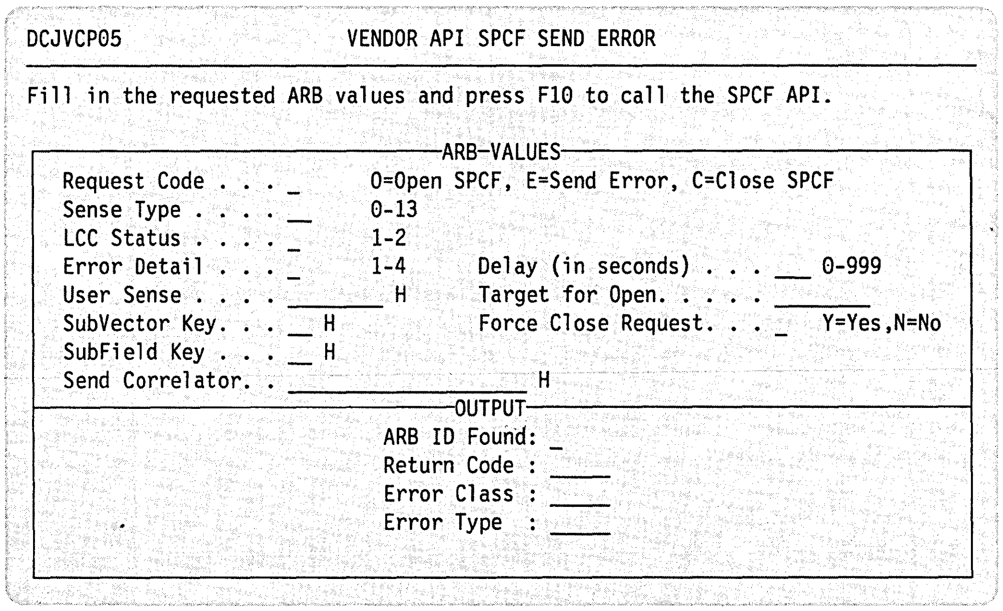
Y Indicates that the API should close even if there are outstanding **Receive** requests.

N Indicates that the API should complete the close successfully only if no **Receive** requests were outstanding.

Output Fields

There are no output fields on this panel, aside from the general output information documented in "Other Information" on page 175.

SPCF Send Error Sense Data



```
DCJVCP05                 VENDOR API SPCF SEND ERROR

Fill in the requested ARB values and press F10 to call the SPCF API.

                              ARB-VALUES
  Request Code . . . _      O=Open SPCF, E=Send Error, C=Close SPCF
  Sense Type . . . . __     0-13
  LCC Status . . . . _      1-2
  Error Detail . . . _      1-4       Delay (in seconds) . . . ___ 0-999
  User Sense . . . . ________ H       Target for Open. . . . . ________
  SubVector Key. . . __ H             Force Close Request. . . _   Y=Yes,N=No
  SubField Key . . . __ H
  Send Correlator. . ____________________ H
                                OUTPUT
                             ARB ID Found: _
                             Return Code : _____
                             Error Class : _____
                             Error Type  : _____
```

Figure 127. SPCF Send Error Panel

This panel is used to send error sense data back to the host. The functions available are:

Function	Description
Open	Open the API. This function must be executed before any other function.
Error	Send error sense data.
Close	Close the API.

Function Keys

The functions keys that are active on this panel, aside from the ones described in "Function Keys" on page 174, are:

Key	Description
F7	Lists the correlators received, and allows selection of one of these active correlators (refer to "SPCF Correlator List" on page 208 for more information).

Parameters

The parameters which must be entered on this panel are:

Request Code = O|E|C
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
E	Issues an ARB with a send request to the API when F10 is pressed. This should result in the error sense data being sent to the host.
C	Issues an ARB with a Close request to the API when F10 is pressed.

Sense Type = *sense_type*
sense_type is a number from 0-13 to be sent to the host. Refer to Figure 15 on page 44 for the meanings of these values.

LCC Status = 1|2
This field is needed if *sense_type* is equal to 3. It contains the status of the Link Connection Component. Refer to "Defined SENSETYPE values" on page 44 for the meanings of these values.

Error Detail = 1|2|3|4
This field is needed if *sense_type* is equal to 3. Refer to "Defined SENSETYPE values" on page 44 for the meanings of these values.

User Sense = *sense_data*
This field is needed if *sense_type* is equal to 0. The user may supply any sense data to NetView, so long as the data conforms to SNA rules.

SubVector key = *sv_key*
sv_key is the hexadecimal key of the subvector which was detected to be in error during a parse.

SubField key = *sf_key*
sf_key is the hexadecimal key of the subfield which was detected to be in error during a parse.

Send Correlator = *hexnum*
Each command received and each reply sent must have a correlator. *hexnum* is a 10-byte hexadecimal number specifying the correlator which is to be sent to the host. This Send Correlator must match an already received Receive Correlator. A list of active correlators may be displayed by pressing F7. Refer to "SPCF Correlator List" on page 208 for information regarding the correlator display panel.

Delay = *delay*
The amount of time to wait before issuing the ARB to the NetView/PC API is determined by *delay*.

Target For Open = *name*
name specifies to the SPCF Router the name by which this application will be known to the NetView host.

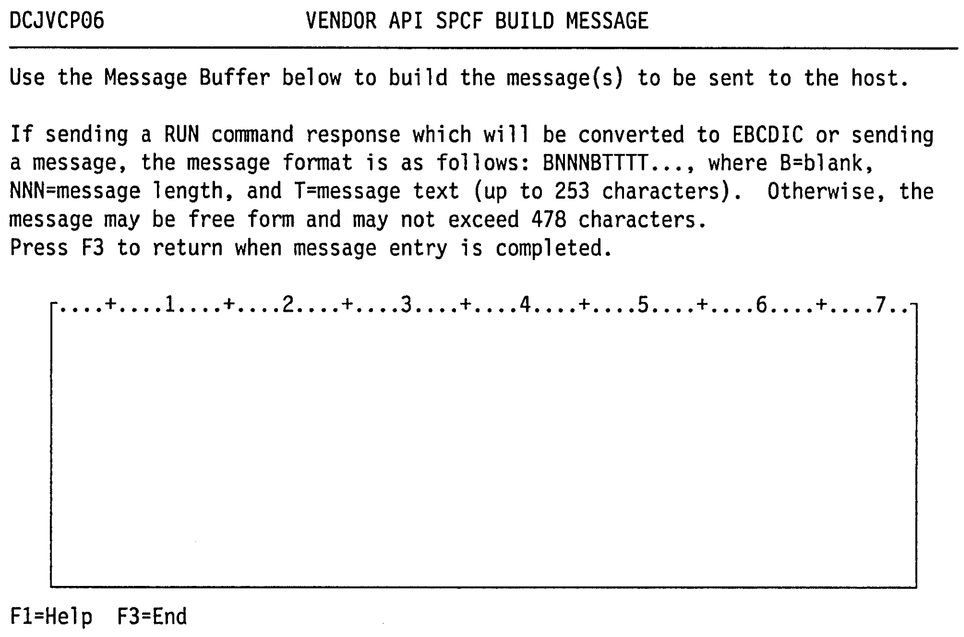
Force Close Request = **Y|N**
This field is meaningful only on a close request (i.e. when **Request Code** is set to **C**).

Y	Indicates that the API should close even if there are outstanding **Receive** requests.
N	Indicates that the API should complete the close successfully only if no **Receive** requests were outstanding.

Output Fields

There are no output fields on this panel, aside from the general output information documented in "Other Information" on page 175.

SPCF Build Message



```
DCJVCP06                    VENDOR API SPCF BUILD MESSAGE

Use the Message Buffer below to build the message(s) to be sent to the host.

If sending a RUN command response which will be converted to EBCDIC or sending
a message, the message format is as follows: BNNNBTTTT..., where B=blank,
NNN=message length, and T=message text (up to 253 characters).  Otherwise, the
message may be free form and may not exceed 478 characters.
Press F3 to return when message entry is completed.

   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..

F1=Help  F3=End
```

Figure 128. SPCF Message Buffer Panel

This panel is used to enter message data for the reply to a RUN command and the SEND MESSAGE unsolicited response panels. Data is entered in the box supplied and will be passed to the API subroutine in a buffer.

Function Keys

There are no active function keys on this panel, aside from those described in "Function Keys" on page 174.

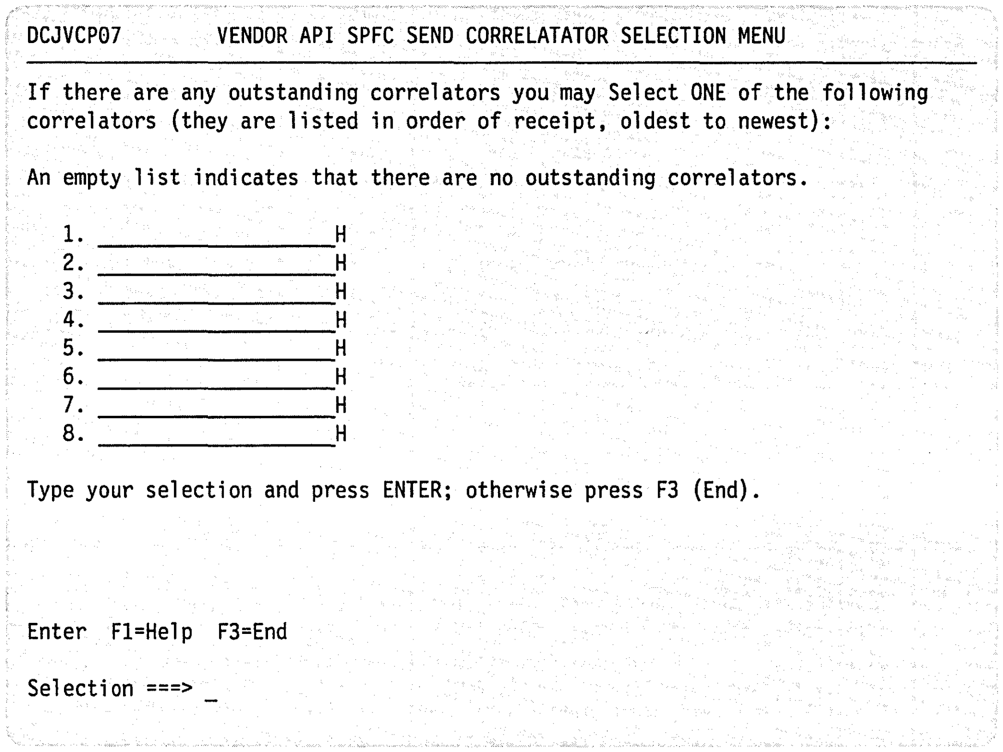
Parameters

The input on this panel is typed in the single input area. It must conform to the instructions given in Figure 128.

Output Fields

There are no output fields on this panel.

SPCF Correlator List



```
DCJVCP07          VENDOR API SPFC SEND CORRELATATOR SELECTION MENU
_______________________________________________________________________

If there are any outstanding correlators you may Select ONE of the following
correlators (they are listed in order of receipt, oldest to newest):

An empty list indicates that there are no outstanding correlators.

   1. ____________________H
   2. ____________________H
   3. ____________________H
   4. ____________________H
   5. ____________________H
   6. ____________________H
   7. ____________________H
   8. ____________________H

Type your selection and press ENTER; otherwise press F3 (End).



Enter  F1=Help  F3=End

Selection ===> _
```

Figure 129. SPCF Correlator Selection Panel

This panel is used to select a Send Correlator to send a RUN command reply, an unformatted reply, or error sense data to the Host.

A selection of 0 will return to the previous panel without making a selection. When a selection is made the user is returned to the previous panel and the Send Correlator field is filled in with the selected Correlator.

When the reply is sent and the return codes are 0 or 8,23,65 (inactivated correlator) the correlator is removed from the list. The ranking of the correlators is last one received, the higher the number selection. Only 8 outstanding GETs are allowed by the SPCF Router.

Function Keys

There are no active function keys on this panel, aside from those described in "Function Keys" on page 174.

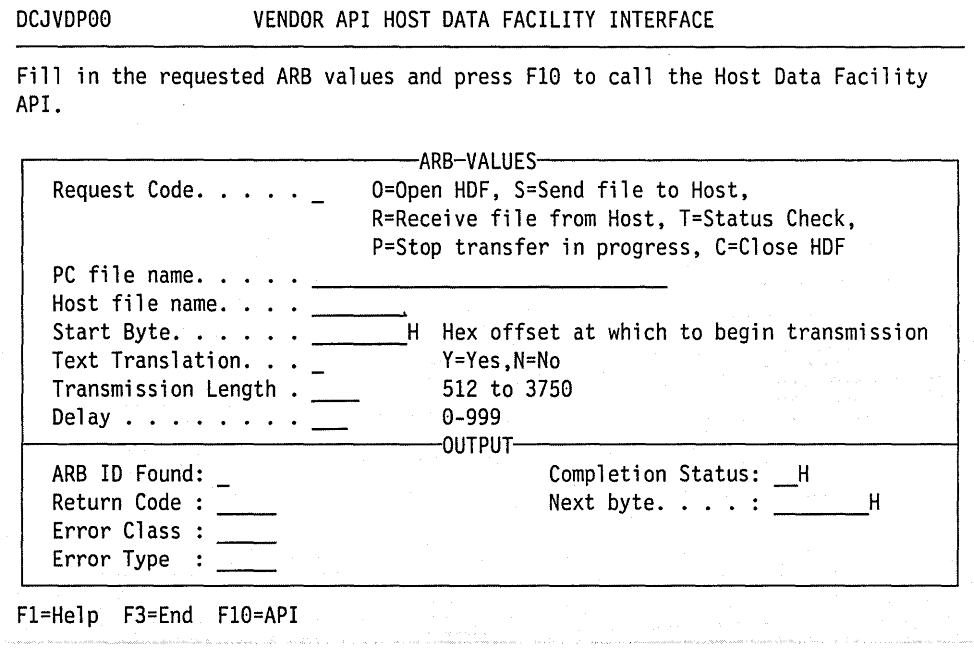
Parameters

Selection = *n*
n is a number between 0 and 8 corresponding to the number of the correlator desired. A value of 0 indicates that no correlator is to be selected.

Output Fields

Correlators	A column of 20-character hexadecimal displays of eight 10-byte correlators.

Host Data Facility Interface



```
DCJVDP00            VENDOR API HOST DATA FACILITY INTERFACE

Fill in the requested ARB values and press F10 to call the Host Data Facility
API.

 ────────────────────────────────ARB-VALUES──────────────────────────────────
   Request Code. . . . . _   O=Open HDF, S=Send file to Host,
                             R=Receive file from Host, T=Status Check,
                             P=Stop transfer in progress, C=Close HDF
   PC file name. . . . . ______________________________
   Host file name. . . . ________
   Start Byte. . . . . . ________H  Hex offset at which to begin transmission
   Text Translation. . . _          Y=Yes,N=No
   Transmission Length . ____       512 to 3750
   Delay . . . . . . . . ___        0-999
 ──────────────────────────────────OUTPUT────────────────────────────────────
   ARB ID Found: _                           Completion Status: __H
   Return Code : _____                       Next byte. . . . : ________H
   Error Class : _____
   Error Type  : _____

F1=Help  F3=End  F10=API
```

Figure 130. Host Data Facility Interface Panel

The Host Data Facility panel is used to exercise the functions provided by the host data facility API. These functions are:

Function	Description
Open	Open the API. This function must be executed before any other function.

Send	Initiate a file transfer to the host from the PC. The application program is then free to perform other tasks while the file transfer is being performed by NetView/PC. The application program may perform a status check at any time after initiating the send to check on the progress of the transfer. Only one file may be sent at a time and a file send and a file receive cannot be performed simultaneously.
Receive	Initiate a file transfer to the PC from the host. The application program is then free to perform other tasks while the file transfer is being performed by NetView/PC. The application program may perform a status check at any time after initiating the receive to check on the progress of the transfer. Only one file may be received at a time and a file receive and a file send cannot be performed simultaneously.
Status	Obtain the status of the file transfer currently in progress or the most recently completed file transfer. Each time a status request is made the **Next Byte** and **Completion Status** fields are returned. See "Output Fields" on page 211 for information on the values of these fields. The status request function does not automatically poll the host data facility. It is the user's responsibility to perform the polling. This is done by periodically pressing the F10 key with **T** in the **Request Code** field. Polling too frequently will slow down or even stop the tranfer of data. For this reason it is recommended that you do a status request no more frequently than every two seconds. Also, be careful not to poll too many times as the completion code of 80 hex will only be returned once. Status requests made after the 80 hex has been returned but before another file transfer is initiated will result in return codes that will indicate that status is not available. (Refer to "Output Fields" on page 211 for more information on the completion codes).
Stop	Stop the transfer in progress.
Close	Close the API.

Function Keys

There are no active function keys on this panel, aside from those described in "Function Keys" on page 174.

Parameters

The parameters which must be entered on this panel are:

Request Code = O|S|R|T|P|C
Determines the type of request to place in the ARB.

O	Issues an ARB with an Open request to the API when F10 is pressed.
S	Issues an ARB with a Send request to the API when F10 is pressed. This should result in the selected file being transferred to the host.
R	Issues an ARB with a Receive request to the API when F10 is pressed. This should result in a file being received from the host.

T	Issues an ARB with a Status request to the API when F10 is pressed. The status of the present file transfer is updated on the screen.
P	Issues an ARB with a Stop request to the API when F10 is pressed. The file transfer in progress is halted.
C	Issues an ARB with a Close request to the API when F10 is pressed.

PC Filename = *filename.ext*
filename.ext is the name (and extension) of a DOS file which is to be transferred to the host (if the request code is **S**), or the name of a file on the PC into which a host file will be received (if the request code is **R**).

Host Filename = *filename*
filename is the eight character name of a file which is to be transferred from the host (if the request code is **R**), or the name of a file on the host into which a PC file will be sent (if the request code is **S**).

Start Byte = *hex_number*
hex_number is the hexadecimal number indicating the offset (in bytes) into a file at which file transfer is to begin. An offset of 0 will begin transferring from the first byte of a file.

Text Translation = **Y|N**
Determines whether translation is to occur.

Y	If a PC file is being sent to the host (request code **S**), it is translated from ASCII to EBCDIC. If a host file is being received onto the PC (request code **R**), it is translated from EBCDIC to ASCII. This option should be selected for readable text files.
N	If this option is selected, no character translation takes place. This option should be used for exchange of files containing binary data.

Transmission Length = *length.*
length is a decimal number indicating the size of the blocks of data transferred between the host and PC.

Delay = *delay*
delay is the amount of time to wait before beginning the file transfer. This is actually the amount of time to wait before issuing the ARB, and is not a field in the ARB itself.

Output Fields

In addition to the standard output fields described in "Other Information" on page 175, the host data facility panel provides the following addition information:

Field	Description
Completion Status	Indicates whether or not the file transfer is currently in progress. This field will contain one of three possible values. A value of 0 indicates that the file transfer is still in progress. A value of 40 hex indicates that the file transfer has been aborted, either due to a Stop request by the user or an error condition that terminated the file transfer. A value of 80 hex indicates that the file tranfer has been successfully completed.

Next Byte	The Next Byte field is an unsigned 32 bit hex number that is one greater than the number of bytes that have been transferred. On completion of the file transfer the Next Byte will return a value one greater than the size of the file being transferred.

Appendix H. API Sample Program Error Messages

DCJV0001E Non Hex character in Request Code. Only 0-9, A-F allowed.

Explanation: An attempt was made to enter a non-hexadecimal value in a hexadecimal input field.

User Response: Correct the input to contain only valid hexadecimal characters, as stated in the message.

DCJV0002E Error Opening Test Case file.

Explanation: The specified test case file did not exist, or an I/O error occurred while attempting to open the file.

User Response: Check to make sure you have entered the correct name, and that the file exists.

DCJV0003E Error Reading Test Case file.

Explanation: An error occurred while the program was trying to obtain data from the test case file.

User Response: Repeat the operation. If it fails again, create a new test case file and repeat again.

DCJV0004E Error Closing Test Case file.

Explanation: An error occurred while closing a test case file. This error should not occur; its presence indicates an operating system or hardware malfunction.

User Response: Repeat the operation. If the operation fails, re-IPL the system.

DCJV0005E Error Opening requested file. Return Code = &IORETCOD.

Explanation: The specified test case file did not exist, or an I/O error occurred while attempting to open the file.

User Response: Check to make sure you have entered the correct name, and that the file exists.

DCJV0006E Error Reading requested file. Return Code = &IORETCOD.

Explanation: An error occurred while the program was trying to obtain data from the file.

User Response: Repeat the operation. If it fails again, create a new file and repeat again.

DCJV0007E Error Closing requested file. Return Code = &IORETCOD.

Explanation: An error occurred while closing a file. This error should not occur; its presence indicates an operating system or hardware malfunction.

User Response: Repeat the operation. If the operation fails, re-IPL the system.

DCJV0008E Invalid choice. Enter 1 - 4 or press F3 to quit.

Explanation: Only four functions are available in the API Sample Program; these functions are numbered from 1 to 4, and only 1 - 4 may be entered in the Selection field.

User Response: Change the selection value to a valid choice between 1 and 4.

DCJV0009E Non Hex character in Start Byte. Only 0-9, A-F allowed.

Explanation: The start byte contains a non-hexadecimal character. Only the characters 0-9 and A-F may be placed in the field.

User Response: Change the start byte to include only valid hex numbers.

DCJV0010E Length of message(s) is invalid. Valid length is 1 - 253.

Explanation: The message buffer contained a message with a length which was not a decimal number between 1 and 253.

User Response: Correct the length.

DCJV0011E Non-numeric character in length field of Message Buffer.

Explanation: A length field in the Message Buffer contained a character which was not a decimal number.

User Response: Correct the length.

DCJV0012E Non Hex character in Send Correlator. Only 0-9, A-F allowed.

Explanation: The send correlator contains a non-hexadecimal character. Only the characters 0-9 and A-F may be placed in the field.

User Response: Change the start byte to include only valid hex numbers.

DCJV0013E Invalid choice. Enter 1 - 8 or Press F3 for main

Explanation: Only eight functions are available from the SPCF main menu; these functions are numbered from 1 to 8, and only 1 - 8 may be entered in the Selection field. Refer to "Service Point Command Facility Interface" on page 181 for more information.

User Response: Change the selection value to a valid choice between 1 and 8.

DCJV0014E Major Subvector Key is not 8061 - 8064. Unable to parse.

Explanation: An attempt was made to parse an NMVT with a major vector key not in the range of 8061-8064 (hexadecimal).

User Response: Correct the present NMVT or read in a valid NMVT.

DCJV0015E Unable to parse APPL NAME from subvector.

Explanation: The application name could not be derived from the NMVT being parsed.

User Response: Correct the present NMVT or read in a valid NMVT.

DCJV0016E Unable to parse Run Command.

Explanation: An error was found in a Run command NMVT.

User Response: Correct the present NMVT or read in a valid NMVT.

DCJV0017E Unable to parse Link Segment List.

Explanation: An error was found in the link segment list portion of an NMVT.

User Response: Correct the present NMVT or read in a valid NMVT.

DCJV0018E Zero length in Link Segment List.

Explanation: A zero length was found in a link segment list.

User Response: Correct the present NMVT or read in a valid NMVT.

DCJV0019E Unable to parse Test Count.

Explanation: The test count could not be derived from the NMVT.

User Response: Correct the present NMVT or read in a valid NMVT.

DCJV0020E More Link Segment List Names than this program allows.

Explanation: The NMVT being parsed contained more names than this program allows.

User Response: Correct the present NMVT or read in a valid NMVT, or fix this program.

DCJV0021E Link Segment List Name longer length than this program allows.

Explanation: The NMVT being parsed contained a name longer than this program allows.

User Response: Correct the present NMVT or read in a valid NMVT, or fix this program.

DCJV0022E Insufficient storage to store current Receive Correlator.

Explanation: There was not enough memory to store the present receive correlator.

User Response: Reduce the number of active receive correlators by sending responses.

DCJV0023I There are no outstanding Receive correlators.

Explanation: No correlators are active.

DCJV0024E Invalid entry - Enter a number 0 - &UNRSPCNT or press F3.

Explanation: The number selected did not correspond to an active correlator.

User Response: Change the selection to a number from 0 - *n*, where *n* is the number of active correlators.

DCJV0025E Unable to delete current Send Correlator from table.

Explanation: An error occurred while attempting to delete the current Send Correlator from the Correlator table. This error should not occur.

DCJV0026E Parsed Target Name length is zero or greater than allowed.

Explanation: The length of a target name was either zero or of a size greater than allowed by the sample program.

User Response: Correct the Target Name.

DCJV0027E Non Hex character in SV Key field. Only 0-9, A-F allowed.

Explanation: An attempt was made to enter a non-hexadecimal character in a hexadecimal input field.

User Response: Correct the input to contain only hexadecimal characters.

DCJV0028E Non Hex character in SF Key field. Only 0-9, A-F allowed.

Explanation: An attempt was made to enter a non-hexadecimal character in a hexadecimal input field.

User Response: Correct the input to contain only hexadecimal characters.

DCJV0029E Non Hex character in User Sense field. Only 0-9, A-F allowed.

Explanation: An attempt was made to enter a non-hexadecimal character in a hexadecimal input field.

User Response: Correct the input to contain only hexadecimal characters.

DCJV0030E Data in file is larger than the buffer in this program.

Explanation: The file specified contains more data than can be placed in the space allocated for it in the sample program.

Programmer Response: Modify the sample program, and increase the buffer sizes.

User Response: Check to make sure the file contains a valid NMVT, and does not contain any extraneous data.

DCJV0031E The Request Code letter entered is unknown.

Explanation: The request code letter specified is not recognized by the sample program as corresponding to a hexadecimal request number in the API.

User Response: Change the request code to one of the valid letters.

DCJV0032E Move file pointer error reading file. Return Code = &IORETCOD.

Explanation: An error occurred while executing a DOS call to move the file pointer to the NMVT file.

User Response: Check to be certain the NMVT file still exists and has not been modified.

DCJV0033I Send correlator &SENDCORR is inactive and has been deleted.

Explanation: The correlator *SENDCORR* has been deactivated.

DCJV0200E Non Hex character in offset field. Only 0-9, A-F are allowed.

Explanation: An attempt was made to type a non-hexadecimal character in the offset field of the unformatted (dump) display.

User Response: Correct the input to contain only hexadecimal characters.

DCJV0201I Now displaying beginning of dump.

Explanation: The unformatted display is now showing data beginning at the first byte in the specified file.

DCJV0202I Now displaying end of dump.

Explanation: The unformatted display is now showing the last data byte in the specified file.

DCJV0203E Offset desired is past the end of the data.

Explanation: An attempt was made to view an offset greater than the length of the data in the file being displayed.

User Response: Use only hexadecimal offsets less than that specified by the length of dump field.

DCJV0204E Formatted dump display not yet available.

Explanation: An attempt was made to display in formatted form an ARB for which formatted display has not been implemented.

User Response: Select a different file.

DCJV0205E Unable to give formatted display; unknown request code in ARB.

Explanation: The request code in the ARB in a file was not recognized as being one of those for which a formatted display is available.

User Response: Select a different file to view or correct the request code.

DCJV0206E Unable to give formatted display; unknown ARB ID in ARB.

Explanation: The ARB ID field in the ARB in a file was not recognized as being one of those for which a formatted display is available.

User Response: Select a different file to view or correct the ARB ID.

DCJV0207E Unable to give formatted display; ARB length was incorrect.

Explanation: The length field ARB in a file was not recognized as being one of those for which a formatted display is available.

User Response: Select a different file to view.

DCJV0208E Unable to give formatted display; parse ID was not recognized.

Explanation: The parse ID field ARB in a file was not recognized as being one of those for which a formatted display is available.

User Response: Select a different file to view.

DCJV0209E Error creating file for output. Return Code = &IORETCOD.

Explanation: The sample program encountered an error while attempting to create to a file.

User Response: Make certain the current directory has enough room for another file to be created. Also, make sure the filename is a valid DOS filename.

DCJV0210E Error writing requested file. Return Code = &IORETCOD.

Explanation: The sample program encountered an error while attempting to write out to a file.

User Response: Select a different filename, check to make sure the current directory is not full, and check to make certain the disk is not full.

DCJV0211I There are no correlators outstanding.

Explanation: No correlators are active, so none are available to be displayed.

DCJV0300E The name of a file containing an NMVT is required.

Explanation: The filename of an NMVT was left blank, or the file specified did not contain an NMVT.

User Response: Enter the name of a file (including extension) containing an NMVT.

DCJV0301E Parse of *nmvtfile.ext* failed. ARB file was not written to disk.

Explanation: An error was detected while attempting to parse the NMVT in *nmvtfile.ext*; therefore the parse output ARB was not written to the ARB file.

User Response: Change the NMVT filename to one containing a valid NMVT.

DCJV0302E Parse routine was unable to read the NMVT file from disk.

Explanation: An error was detected while attempting to read the selected NMVT file from disk.

User Response: Check the filename to be certain it is the name of an existing file in the current directory containing an NMVT.

DCJV0303E Parse routine was unable to write the ARB file to disk.

Explanation: An error was detected while attempting to save a parse output ARB to the selected disk file.

User Response: Make sure the ARB output filename is valid, and that there is sufficient storage in the current directory for the output file.

DCJV0304E Build API not called: ARB ID in file not equal ARB5.

Explanation: The ARB ID field in the ARB in the selected field was not ARB5. Therefore no call was made to the build API.

User Response: Change the filename or correct the ARB in the file so that the ARB ID is ARB5.

DCJV0409E Build API not called: unknown ARB ID in ARB.

Explanation: The sample program could not create a build request in memory because the ARB ID in the ARB read in from the specified file was not recognized as being valid.

User Response: Select a different file or correct the data in the file.

DCJV0410E Build API not called: unknown request code in ARB.

Explanation: The sample program could not create a build request in memory because the request code in the ARB read in from the specified file was not recognized as being valid.

User Response: Select a different file or correct the data in the file.

DCJV0411E Build API not called: ARB length was incorrect.

Explanation: The sample program could not create a build request in memory because the length field in the ARB read in from the specified file was not recognized as being valid.

User Response: Select a different file or correct the data in the file.

DCJV0412E Build API not called: build ID was not recognized.

Explanation: The sample program could not create a build request in memory because the build ID field in the ARB read in from the specified file was not recognized as being valid.

User Response: Select a different file or correct the data in the file.

DCJV0413E Build API not called: error in the probable cause pointer.

Explanation: The sample program could not create a build request in memory because the pointer to the probable cause code points was not valid (it pointed beyond the data read in from the file).

User Response: Select a different file or correct the data in the file.

DCJV0414E Build API not called: error in the path list info pointer.

Explanation: The sample program could not create a build request in memory because the pointer to the path list information was not valid (it pointed beyond the data read in from the file).

User Response: Select a different file or correct the data in the file.

DCJV0415E Build API not called: error in the path list blocks pointer.

Explanation: The sample program could not create a build request in memory because the pointer to the path list block array was not valid (it pointed beyond the data read in from the file).

User Response: Select a different file or correct the data in the file.

DCJV0416E Build API not called: error in LCC description block pointer.

Explanation: The sample program could not create a build request in memory because the pointer to the LCC description block was not valid (it pointed beyond the data read in from the file).

User Response: Select a different file or correct the data in the file.

DCJV0417E Build API not called: error in LCC data block pointer.

Explanation: The sample program could not create a build request in memory because the pointer to the LCC data block was not valid (it pointed beyond the data read in from the file).

User Response: Select a different file or correct the data in the file.

DCJV0418E An error was detected by the build API.

Explanation: An unspecified error was detected by the build API.

Programmer Response: Debug the sample program to find the cause of the error.

User Response: Select a different file or correct the data in the file.

DCJV0419E There are No LCC's in the ARB file specified.

Explanation: The specified file contained no LCC blocks; there must be at least one.

User Response: Correct the data file.

Appendix I. DOS Sample Program Code

APIMAIN.DSG

```
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.

;*******************************************************************
;*                                                                 *
;*  Program Name : APISAMPL                                        *
;*                                                                 *
;*  Description  : Sample program to allow the interactive creation *
;*                 of ARB's using EZ-VU II panels for user input    *
;*                 and the submission of those ARB's to the API/CS  *
;*                 for execution.                                   *
;*                                                                 *
;*  Date          :  July, 1986; May 1987                          *
;*                                                                 *
;*  Input         :  By entering ARB data into EZ-VU II panels the *
;*                   user may define and execute any of the        *
;*                   functions available through the API/CS.       *
;*                                                                 *
;*                   The only external data used by this program are *
;*                   the NMVT files used by the API/CS Alert       *
;*                   function and the ARB Build data files used as *
;*                   input to the Build routines for SPCF. These   *
;*                   files were created by standalone programs     *
;*                   written in MASM 2.0.                          *
;*                                                                 *
;*  Output        :  For each test case resulting return code(s),  *
;*                   error class(es) and error type(s) and other   *
;*                   output from the API, such as file offset and  *
;*                   completion byte for the Host Data Facility and *
;*                   host command and receive correlator for the   *
;*                   Service Point Command Facility are displayed on *
;*                   the panels.                                   *
;*                                                                 *
;*  Program Type   :  IBM Macro Assembler version 2.0              *
;*                                                                 *
;*  Processor type :  Intel 8088/80286                             *
;*                                                                 *
;*  External references :        Entry points for:                 *
;*                     DCJVA00  -  ALERTS                          *
;*                     DCJVO00  -  OPERATOR COMMUNICATIONS         *
;*                     DCJVC00  -  SERVICE POINT COMMAND FACILITY  *
;*                     DCJVB00  -  BUILD AND PARSE ROUTINES        *
;*                    DCJVD00  -  HOST DATA FACILTY                *
;*                                                                 *
;*                    ISPASMV  -  EZ-VU II Variable Definitions    *
;*                    ISPASM   -  EZ-VU II Display Functions       *
;*******************************************************************
PAGE

DGROUP  GROUP   DATA,STACK

STACK   SEGMENT BYTE STACK 'STACK'
        DB      256 DUP('STACK   ')      ; 2K STACK AREA
STKTOP  DW      1
STACK   ENDS

DATA    SEGMENT PARA PUBLIC 'DATA'
        ASSUME  DS:DGROUP


        CR_LF EQU WORD PTR 0A0DH      ; ASCII Code for Carriage Return/Line Feed
        CR    EQU BYTE PTR 0DH        ; ASCII Code for Carriage Return
        LF    EQU BYTE PTR 0AH        ; ASCII Code for Line Feed
        ESC   EQU BYTE PTR 27D        ; ASCII Code for Escape Code
        F1    EQU BYTE PTR 59D        ; Scan Code for F1  key
        F2    EQU BYTE PTR 60D        ; Scan Code for F2  key
        F3    EQU BYTE PTR 61D        ; Scan Code for F3  key
        F4    EQU BYTE PTR 62D        ; Scan Code for F4  key
        F5    EQU BYTE PTR 63D        ; Scan Code for F5  key
        F6    EQU BYTE PTR 64D        ; Scan Code for F6  key
        F7    EQU BYTE PTR 65D        ; Scan Code for F7  key
        F8    EQU BYTE PTR 66D        ; Scan Code for F8  key
        F9    EQU BYTE PTR 67D        ; Scan Code for F9  key
        F10   EQU BYTE PTR 68D        ; Scan Code for F10 key


        PAGEUP EQU BYTE PTR 73d         ;Page up scan code
        PAGEDN EQU BYTE PTR 81d         ;Page down scan code


ioretcod   DW   0                     ;File I/O return code for error messages.

EXITFLAG   DW   0                     ; FLAG, ON INDICATES A PANEL WAS
                                      ; EXITED USING THE F2-QUIT KEY

AX_REG     DW   0                     ; Save area for AX:DX regs for use
DX_REG     DW   0                     ; in checking to assure that the
                                      ; that the API/CS actually found
                                      ; the ARB passed to it. Used by
                                      ; subrout CHECK_ARB

ARB_FOUND  DB   ' '                   ; Set to Y if ARB was found
                                      ; by call to any API/CS function.
                                      ; Set to N if not. Used by
                                      ; subrout CHECK_ARB.

PAGE
```

```
;*
;* ALERT VARIABLES
;*

ALERT_RC_TBL EQU $                      ; Alert Request Code Lookup Table
             DB 'O'                     ; Open Alerts
             DW  0101H
             DB 'S'                     ; Send an Alert
             DW  0102H
             DB 'C'                     ; Close Alerts
             DW  0104H
             DB '*'                     ; End of Table Marker

DISPTYPE       DB  ' '                  ; Type of display required:
                                        ;   A = ASCII
                                        ;   E = EBCDIC
                                        ;   F = Formatted

DELAY1         DW  0                    ; Number of seconds to wait before
                                        ; calling the Alert API/CS to
                                        ; execute the Alert ARB

ALERT_VISITED  DB  0                    ; 0 indicates Alert Panel has
                                        ; never been visited. 1 indicates
                                        ; that it has been visited.

ARB_FOUND1 DB   ' '                     ; Set to Y if ARB was found
                                        ; by call to Alert API/CS.
                                        ; Set to N if not.


REQCODE1_ASC DB 'O'                     ; Input buffer for ASCII form
                                        ; of Alert Request Code. Is
                                        ; converted to HEX and stored
                                        ; in REQ_CODE1 by subrout
                                        ; GET_REQCODE
PAGE
;*
;* ALERT ARB
;*
ARB_ID1   DB  'ARB1'                    ; Alert ARB ID
REQ_CODE1 DW  0101H                     ; Alert request code
ARB_LNG1  DB    44D                     ; Length of the Alert ARB

RESRV1_1  DW     0H                     ; reserved word
PRIME_RC1 DW 0FFFFH                     ; Alert primary Return Code
PRIME_EC1 DW 0FFFFH                     ; Alert primary Error Class
PRIME_ET1 DW 0FFFFH                     ; Alert primary Error Type
NMVTADDR  DD NMVTBUFF                   ; Address of the buffer containing
                                        ; the Alert NMVT to be sent.
NMVTTARG  DB    'L'                     ; Alert NMVT target L, H or B
                                        ; Local, Host or Both. Defaults
                                        ; to B if not L, H or B
ALERT_RC1 DW 0FFFFH                     ; Alert manager Return Code
ALERT_EC1 DW 0FFFFH                     ; Alert manager Error Class
ALERT_ET1 DW 0FFFFH                     ; Alert manager Error type
CSSA_RC1  DW 0FFFFH                     ; CSSA Return Code
CSSA_EC1  DW 0FFFFH                     ; CSSA Error Class
CSSA_ET1  DW 0FFFFH                     ; CSSA Error Type
```

```
HOST_RC1  DW 0FFFFH                ; Host Return Code
HOST_EC1  DW 0FFFFH                ; Host Error Class
HOST_ET1  DW 0FFFFH                ; Host Error Type
RESERV2_1 DW 3 DUP(0H)             ; 3 reserved words

;*
;* END OF ALERT ARB
;*

PAGE
;*
;* OPERATOR COMMUNICATIONS VARIABLES
;*

OPCOMM_RC_TBL EQU $                ; Operator Communications Request Code Lookup Table
          DB 'O'                   ; Open Operator Communications
          DW  0201H
          DB 'W'                   ; Write the icon DP
          DW  0207H
          DB 'L'                   ; Clear the icon DP
          DW  0208H
          DB 'C'                   ; Close Operator Communications
          DW  0204H
          DB '*'                   ; End of Table Marker

DELAY2        DW  0                ; Number of seconds to wait before
                                   ; calling the Op Comm API/CS to
                                   ; execute the Op Comm ARB

OPCOMM_VISITED  DB  0              ; 0 indicates Op Comm Panel has
                                   ; never been visited. 1 indicates
                                   ; that it has been visited.

ARB_FOUND2 DB   ' '                ; Set to Y if ARB was found
                                   ; by call to Op Comm API/CS.
                                   ; Set to N if not.

REQCODE2_ASC   DB 'O'              ; Input buffer for ASCII form
                                   ; of Oper Comm Request Code. Is
                                   ; converted to HEX and stored
                                   ; in REQ_CODE2 by subrout GET_REQCODE

PAGE
;*
;* OPERATOR COMMUNICATIONS ARB
;*
ARB_ID2   DB  'ARB2'
REQ_CODE2 DW  0201H
ARB_LNG2  DB    15D

RESRV1_2  DW     0H
PRIME_RC2 DW 0FFFFH
PRIME_EC2 DW 0FFFFH
PRIME_ET2 DW 0FFFFH
;*
;* END OF OPERATOR COMMUNICATIONS ARB
;*
```

```
PAGE
;*
;* SERVICE POINT COMMAND FACILITY VARIABLES
;*

SPCF_RC_TBL EQU $                    ; SPCF Request Code Lookup Table
            DB 'O'                   ; Open SPCF
            DW  0301H
            DB 'S'                   ; Send a response to a command from NetView
            DW  0302H
            DB 'R'                   ; Receive a command from NetView
            DW  0303H
            DB 'C'                   ; Close SPCF
            DW  0304H
            DB 'G'                   ; Get No Parse
            DW  0309H
            DB 'M'                   ; Send Message Unsolicited
            DW  030AH
            DB 'P'                   ; Put Message Unformatted
            DW  030BH
            DB 'E'                   ; Send Error Sense Codes
            DW  030CH
            DB '*'                   ; End of Table Marker


DELAY3         DW  0                 ; Number of seconds to wait before
                                     ; calling the SPCF API/CS to
                                     ; execute the SPCF ARB

SPCF_VISITED   DB  0                 ; 0 indicates SPCF Panel has
                                     ; never been visited. 1 indicates
                                     ; that it has been visited.

ARB_FOUND3 DB   ' '                  ; Set to Y if ARB was found
                                     ; by call to SPCF API/CS.
                                     ; Set to N if not.

REQCODE3_ASC   DB 'O'                ; Input buffer for ASCII form
                                     ; of SPCF Request Code. Is
                                     ; converted to HEX and stored
                                     ; in REQ_CODE3 by subrout GET_REQCODE

MSGBUFFR1       DB  630 DUP (' ')    ; Input buffer for message(s) to
                                     ; to be sent to the host.
MSGBUFFR2       DB  630 DUP (' ')    ; Buffer used to build multiple
                                     ; message block for passing
                                     ; multiple messages to the host.

PAGE

COMMAND         DB  512 DUP (' ')    ; Output buffer for displaying
                                     ; command received from the host.
ASC_CORR_LENGTH     EQU 20
MAX_CORR_CNT        EQU  8
CORR_ASC_TBL_LENGTH EQU ASC_CORR_LENGTH * MAX_CORR_CNT
BY_ASC_CORR_LENGTH  DW  ASC_CORR_LENGTH

CORR_DELETED    DB   0        ; Save area for DEL_SENDCORR
```

```
CORR_ASC_TBL    DB   MAX_CORR_CNT DUP(ASC_CORR_LENGTH DUP(' '))
CORR_RANK_TBL   DB   MAX_CORR_CNT DUP(0FFH)

RECVCORR_HEXASC DB ASC_CORR_LENGTH DUP ('0') ; Output buffer for HEX/ASCII
                                             ; form of SPCF receive
                                             ; correlator.

SENDCORR_HEXASC DB ASC_CORR_LENGTH DUP ('0') ; Input buffer for HEX/ASCII
                                             ; form of SPCF send correlator

TEMPCORR_HEXASC DB ASC_CORR_LENGTH DUP ('0') ; Temp buffer for HEX/ASCII
                                             ; correlator while sorting

CORROPT         DB 0                  ; Correlator chosen in LOAD_SENDCORR

UNRESPONDED_CNT DB 0                  ; Number of outstanding replies

SORTFLAG        DB 1                  ; Sort termination flag used when
                                      ; sorting correlator table in
                                      ; LOAD_SENDCORR.

SENDCORR_STAT   DB  0                 ; FF hex indicates conversion of
                                      ; SENDCORR_ASC to SENDCORR by
                                      ; subrout CNV_SENDCORR was
                                      ; unsuccessful. 0 indicates
                                      ; successful conversion.

INACT_CORR      DB  0                 ; Flag on indicates correlator
                                      ; has been inactivated.

SENDCORR_CNT    DB  0                 ; Used as count variable from
                                      ; 5 to 1 for converting each
                                      ; of the 5 hex words in SENDCORR

LOADSTAT        DB  0                 ; FF hex indicates that Loading of
                                      ; the Message Buffer by
                                      ; subrout LOAD_MSGBUFF was
                                      ; unsuccessful. 0 indicates
                                      ; successful conversion.

CURRMSG_NUM     DW  0                 ; Contains the number of the
                                      ; message in the multi-message
                                      ; buffer currently being loaded.
PUTREPLY_BUFF_SIZE EQU 512
PUTREPLY_LNG EQU $                    ; Length field of SPCF NMVT read from file
PUTREPLY_KEY EQU $ + 2                ; Key    field of SPCF NMVT read from file
                                      ; Buffer area for SPCF NMVT
PUTREPLY     DB PUTREPLY_BUFF_SIZE/8 DUP ('SPCFPUT ')

USERSENSE_ASC DB '00000000'           ; ASCII Input buffer for USERSENSE

SVKEY_ASC_PRE DB '00'                 ; Leading ASCII zeroes because the subrout
                                      ; ASC2HEX expects a 4 byte string.
SVKEY_ASC     DB '00'                 ; ASCII Input buffer for SVKEY

SFKEY_ASC_PRE DB '00'                 ; Leading ASCII zeroes because the subrout
                                      ; ASC2HEX expects a 4 byte string.
SFKEY_ASC     DB '00'                 ; ASCII Input buffer for SFKEY
```

```
SPCFOPT       DB 1                   ; Option variable for SPCF Menu

NMVTNAME      DB 'NMVTFILE.BIN '     ; File name of binary image file
                                     ;   for Put Unformatted SPCF

RECID_ASC     DB 2  DUP (' ')        ; Hex ASCII form of RECID

PAGE
;*
;* SERVICE POINT COMMAND FACILITY ARB
;*
ARB_ID3     DB  'ARB3'
REQ_CODE3   DW  0301H
ARB_LNG3    DB    90D

RECID       DB     0H
RESRV1_3    DB     0H

PRIME_RC3   DW 0FFFFH
PRIME_EC3   DW 0FFFFH
PRIME_ET3   DW 0FFFFH

TARGNAME    DB 'TS1SPCI '
MSGTYPE     DB 'F'
MSGFILE     DB 'SPCF'
MSGNUM      DB '0001'
MBLENGTH    DW 0
MSGCOUNT    DW 0
CONVERT     DB 'Y'
MSGBUFF_PTR DD MSGBUFFR1
CMDLGTH     DB 0
COMMAND_PTR DD COMMAND
RECVCORR    DB 10 DUP (0)
SENDCORR    DB 10 DUP (0)
FORCE       DB 'Y'
OPERNAME    DB 'OPERATOR'
PUTRPLY_LEN DW ?
PUTRPLY_PTR DD PUTREPLY
SENSETYP    DB 1
LCCSTAT     DB 0
ERRDETAL    DB 0
USERSENSE   DB 4 DUP(0)
SVKEY       DB 0
SFKEY       DB 0
;*
;* END OF SERVICE POINT COMMAND FACILITY ARB
;*

PAGE
;*
;* HOST DATA TRANSFER FACILITY VARIABLES
;*

HDF_RC_TBL EQU $                     ; HDF Request Code Lookup Table
             DB 'O'                  ; Open HDF
             DW  0401H
             DB 'S'                  ; Send a file from PC to CICS
             DW  0402H
             DB 'R'                  ; Recieve a file from CICS
```

```
            DW  0403H
            DB 'T'                    ; Check status of HDF transfer request
            DW  0405H
            DB 'P'                    ; Stop a file tranfer request
            DW  0406H
            DB 'C'                    ; Close HDF
            DW  0404H
            DB '*'                    ; End of Table Marker

DELAY4        DW  0                   ; Number of seconds to wait before
                                      ; calling the HDF API/CS to
                                      ; execute the HDF ARB

HDF_VISITED   DB  0                   ; 0 indicates HDF Panel has
                                      ; never been visited. 1 indicates
                                      ; that it has been visited.

ARB_FOUND4 DB   ' '                   ; Set to Y if ARB was found
                                      ; by call to HDF API/CS.
                                      ; Set to N if not.

REQCODE4_ASC   DB '0'                 ; Input buffer for ASCII form
                                      ; of HDF Request Code. Is
                                      ; converted to HEX and stored
                                      ; in REQ_CODE4 by subrout GET_REQCODE

STARTBYTE_ASC DB '00000000'           ; Input buffer for HEX/ASCII form
                                      ; STARTBYTE

NEXTBYTE_ASC  DB '00000000'           ; Output buffer for HEX/ASCII form
                                      ; of NEXTBYTE

XFERCOMP_ASC  DB '00  '               ; Output buffer for HEX/ASCII form
                                      ; of XFERCOMP. Only leftmost 2
                                      ; bytes are used, but buffer is
                                      ; 4 bytes long because subrout
                                      ; HEX2ASC expects a 4 byte output
                                      ; buffer.

PAGE

PCFILENM      DB 32 DUP(' ')          ; Name of the PC file to be sent
                                      ; or received. Variable is one byte
                                      ; longer than max filename length
                                      ; so that you can always find the
                                      ; end of the filename by searching
                                      ; for a blank

HOSTFILENM    DB 9  DUP(' ')          ; Name of the Host file to be sent
                                      ; or received. Variable is one byte
                                      ; longer than max filename length
                                      ; so that you can always find the
                                      ; end of the filename by searching
                                      ; for a blank

PAGE
;*
;* HOST DATA TRANSFER FACILITY ARB
;*
```

```
ARB_ID4   DB  'ARB4'
REQ_CODE4 DW  0401H
ARB_LNG4  DB    45D

RESRV1_4  DW     0H
PRIME_RC4 DW 0FFFFH
PRIME_EC4 DW 0FFFFH
PRIME_ET4 DW 0FFFFH
PCFILE    DD PCFILENM
PCFLGTH   DB     0H
HOSTFILE  DD HOSTFILENM
HFLGTH    DB     0H
STARTBYTE DD     0H
XPC       DB    'N'
BLKZ      DW  3750D
RESERV2_4 DB 8 DUP(0H)
NEXTBYTE  DD     0H
XFERCOMP  DB     0H

;*
;* END OF HOST DATA TRANSFER FACILITY ARB
;*

PAGE
ARB_FOUND5 DB   ' '                    ; Set to Y if ARB was found
                                       ; by call to Op Comm API/CS.
                                       ; Set to N if not.

;*
;* SERVICE POINT COMMAND FACILITY BUILD ARB
;*
ARB_ID5     DB  'ARB5'
REQ_CODE5   DW  0000H
ARB_LNG5    DB    37D

BUILD_ID    DB     0H
BLD_RESERVE DB     0H

PRIME_RC5   DW 0FFFFH
PRIME_EC5   DW 0FFFFH
PRIME_ET5   DW 0FFFFH

BUILD_NMVT_PTR    DD 0 ;;;BUILD_NMVT
BUILD_NMVT_LEN    DW 0
PATH_LIST_PTR     DD 0 ;;;PATH_LIST
LINK_STATUS       DB 0
NO_PROB_CAUSES    DB 0
PROB_CAUSES_PTR   DD 0 ;;;PROB_CAUSES
LINK_TEST_RESULTS DB 0
TEST_TYPE         DB 0
TEST_COUNT_REQ    DW 0
TEST_COUNT_EX     DW 0
;*
;* END OF SERVICE POINT COMMAND FACILITY ARB
;*


PAGE
ARB_FOUND6 DB   ' '                    ; Set to Y if ARB was found
```

```
                                          ; by call to Op Comm API/CS.
                                          ; Set to N if not.

PAGE

;*
;* SERVICE POINT COMMAND FACILITY PARSE ARB
;*

ARB_ID6           DB  'ARB6'
REQ_CODE6         DW  0000H
ARB_LNG6          DB    36D

PARSE_ID          DB     0H
PARSE_RESERVED    DB     0H

PRIME_RC6         DW 0FFFFH
PRIME_EC6         DW 0FFFFH
PRIME_ET6         DW 0FFFFH

PARSE_NMVT_PTR    EQU  $
PARSE_NMVT_OFFSET  DW 0
PARSE_NMVT_SEGMENT DW 0

NO_NAMES          DB 0
NAMES_PTR         DD 0
TEST_COUNT        DW 0
TEST_TYPE6        DB 0

PARSE_SENSE_DATA_LEN  EQU 4
PARSE_SENSE_ASCII_LEN EQU PARSE_SENSE_DATA_LEN * 2

PARSE_SENSE_DATA   DB PARSE_SENSE_DATA_LEN DUP(0)
PARSE_COMMAND_LEN  DB 0
PARSE_COMMAND_PTR  DD 0

;*
;* END OF SERVICE POINT COMMAND FACILITY PARSE ARB
;*
PARSE_CORRELATOR   DB 10 dup(0)
PARSE_DATA  DB 512 DUP (0H)
PARSE_DATA_OFFSET  EQU PARSE_DATA-ARB_ID6     ;Offset to parse data
DO_PARSE_RC        DB  0                      ;Return code for do_parse proc

PARSE_SENSE_ASCII  DB  PARSE_SENSE_ASCII_LEN DUP('0')   ; Display buffer for
                                                        ; ASCII representation
                                                        ; of PARSE_SENSE_DATA
;*
;* The following are for the build panel
;*
bd_rx           dw      0               ;Return code
bd_bufsize      equ     NMVTBUFF_SIZE   ;Buffer size

;*******************************************************************
; Build ARB Structure definition
;*******************************************************************

build_arb       struc
bd_arbid        db      'ARB5'
```

```
bd_reqcode      dw      0h
bd_arblen       db      37
bd_buildid      db      0
bd_reserved     db      0
bd_retcode      dw      0
bd_errclass     dw      0
bd_errtype      dw      0
bd_builtnmvt    dd      0
bd_builtnmvtlen dw      0
bd_pathlist     dd      0
bd_linkstat     db      0
bd_numprobcause db      0
bd_probcause    dd      0
bd_testresults  db      0
bd_testtype     db      0
bd_testcountreq dw      0
bd_testcountex  dw      0
bd_correlator   db      10      dup(?)          ;Correlator
bd_builddata    db      0                       ;Start of data
build_arb       ends

bd_buf          equ     Arbbuff                 ;Buffer for input

bd_refarb       build_arb <>                    ;Reference ARB

;*********************************************************************
; Now the various path information structures
;*********************************************************************

;Path Information List Control Block
bd_pinfo        struc
bdlcc_num       dw      0               ;Number of LCC resources
bdlcc_ptr       dd      0               ;Pointer to first LCC block
bd_pinfo        ends

;LCC description control blocks
bd_lccdesc      struc
bdlcc_typelen   db      0               ;Length of resource type
bdlcc_type      db      8 dup(?)        ;Resource type
bdlcc_namelen   db      0               ;Length of resource name
bdlcc_name      db      8 dup(?)        ;Resource name
bdlcc_number    dw      0               ;Number of LCC data things
bdlcc_dataptr   dd      0               ;Pointer to LCC data
bdlcc_end       db      0
bd_lccdesc      ends


;LCC data control block structure
bdlccdat        struc
bdlcc_dvtype    db      0               ;Data value type
bdlcc_dvlen     db      0               ;Data value length
bdlcc_reserved  db      0               ;Reserved
bdlcc_dvptr     dd      0               ;Pointer to data value
bdlcc_dnlen     db      0               ;Data name length
bdlcc_dn        db      0               ;Data name
bdlcc_dvend     db      0               ;Data name
bdlccdat        ends
```

```
;*********************************************************************
;Error numbers
;*********************************************************************
BD_NOERR        EQU     0               ;No error
BD_READERR      EQU     0ffffh          ;Error reading file
BD_ARBIDERR     EQU     409d            ;Error in ARB ID
BD_REQCDERR     EQU     410d            ;Error in ARB request code
BD_ARBLENERR    EQU     411d            ;Error in ARB length
BD_ARBBLDIDERR  EQU     412d            ;Error in build ID

BD_PCAUSERR     EQU     413d            ;Error in Probable cause pointer
BD_PLISTERR     EQU     414d            ;Error in Path List Info pointer
BD_PLCBERR      EQU     415d            ;Error in Path List Blocks pointer
BD_LCCDCERR     EQU     416d            ;Error in LCC description block
BD_LCCDBERR     EQU     417d            ;Error in LCC data  block
BD_BLDERR       EQU     418d            ;Error after calling build API
BD_NOLCC        EQU     419d            ;No LCC's in ARB

;*********************************************************************
;Other constants
;*********************************************************************
BD_ARBIDLEN     EQU     4               ;Length of ARB ID
BD_FILENAMELEN  EQU     13              ;Length of file name

BD_ID_LPD       EQU     62h             ;Build ID for LINK PD
BD_ID_LD        EQU     63h             ;Build ID for LINK DATA
BD_ID_LT        EQU     64h             ;Build ID for LINK TEST

BD_TYPE0        EQU     0               ;LINK PD type
BD_TYPE1        EQU     1               ;LINKDATA or LINKTEST type

BD_LCCDCLEN0    EQU     bdlcc_number-bdlcc_typelen      ;Length of type 0
BD_LCCDCLEN1    EQU     bdlcc_end-bdlcc_typelen         ;Length of type 1
BD_LCCDBLEN     EQU     bdlcc_dn-bdlcc_dvtype           ;Length of data

;*********************************************************************
; Working variables
;
bd_type         db      0               ;Type of build: 0 = LINK PD
                                        ;   Nonzero indicates DATA or TEST
bd_lccdclen     dw      0               ;Used to store size of lcc dc

bd_datsize      dw      0               ;Amount of data in buffer


PAGE

CHOICE   DW 1             ; State variable for SELMENU

ZRSP1    DB ?             ; Scan  code of key that caused Panel Exit
ZRSP2    DB ?             ; ASCII code of key that caused Panel Exit

;*********************************************************************
;If number of places below changes, must also change LNGTH9V
;*********************************************************************
ZENT1    DB 0             ; Scan code of key to be used as Enter key
ZENT2    DB 0             ; ASCII code of key to be used as Enter key
ZENT1a   DB 0             ; Scan code of key to be used as Enter key
ZENT2a   DB 0             ; ASCII code of key to be used as Enter key
```

```
ZENT1b   DB 0            ; Scan code of key to be used as Enter key
ZENT2b   DB 0            ; ASCII code of key to be used as Enter key
ZENT1c   DB 0            ; Scan code of key to be used as Enter key
ZENT2c   DB 0            ; ASCII code of key to be used as Enter key
ZENT1d   DB 0            ; Scan code of key to be used as Enter key
ZENT2d   DB 0            ; ASCII code of key to be used as Enter key

ZENT1E   DB 1D           ; Scan code of ESC key
ZENT2E   DB 27D          ; ASCII code of ESC key

ZENT1PUP  DB 73d          ; Scan code of PgUp key
ZENT2PUP  DB 0            ; ASCII code of PgUp key

ZENT1PDN  DB 81d          ; Scan code of PgUp key
ZENT2PDN  DB 0            ; ASCII code of PgUp key

ZENT1F   DB 68D          ; F10 key - scan  code
ZENT2F   DB 0            ; F10 key - ASCII code

ZENT1N   DB 1CH          ; Return key - scan  code
ZENT2N   DB 13D          ; Return key - ASCII code

ZATR     DB 'EW EW'      ; Color used when input field is highlighted
                         ; Ebony foreground, white background

PARM999    DW  7541H
           DW  6874H
           DW  726FH
           DW  203AH
           DW  694AH
           DW  206DH
           DW  6F46H
           DW  6C77H
           DW  7265H
           DW  3320H
           DW  322FH
           DW  0D38H
           DW  1A0AH

PAGE
;*
;* Alert Variable Definitions
;*
PARM2    DB    'REQCODE1 C'
LNGTH2P  DW    LNGTH2P - PARM2
LNGTH2V  DW    1

PARM4    DB    'NMVTFILE C'
LNGTH4P  DW    LNGTH4P - PARM4
LNGTH4V  DW    13

PARM5    DB    'NMVTTARG C'
LNGTH5P  DW    LNGTH5P - PARM5
LNGTH5V  DW    1

PARM6    DB    '(PRIMRC1,PRIMEC1,PRIMET1) I'
LNGTH6P  DW    LNGTH6P - PARM6
LNGTH6V  DW    2
```

```
PARM7    DB    '(ALRTRC1,ALRTEC1,ALRTET1,CSSARC1,CSSAEC1,CSSAET1,HOSTRC1,HOSTEC1,HOSTET1) I'
LNGTH7P  DW    LNGTH7P - PARM7
LNGTH7V  DW    2

PARM8    DB    'ZRSP C'
LNGTH8P  DW    LNGTH8P - PARM8
LNGTH8V  DW    2

PARM9    DB    'ZENT C'
LNGTH9P  DW    LNGTH9P - PARM9
LNGTH9V  DW    10d                ;Must be set to number of places for
                                  ;Other keys being used as enter key

PAGE

PARM10   DB    'CHOICE I'
LNGTH10P DW    LNGTH10P - PARM10
LNGTH10V DW    2

PARM46    DB    'ARBFND1 C'
LNGTH46P  DW    LNGTH46P - PARM46
LNGTH46V  DW    1

PARM50    DB    'DELAY1 I'
LNGTH50P  DW    LNGTH50P - PARM50
LNGTH50V  DW    2

PAGE
;*
;* Operator Communications Variable Definitions
;*
PARM12    DB    'REQCODE2 C'
LNGTH12P  DW    LNGTH12P - PARM12
LNGTH12V  DW    1

PARM14   DB    '(PRIMRC2,PRIMEC2,PRIMET2) I'
LNGTH14P  DW    LNGTH14P - PARM14
LNGTH14V  DW    2

PARM15   DB    'ZATR C'
LNGTH15P DW    LNGTH15P - PARM15
LNGTH15V DW    5

PARM16    DB    'ZFLD C'
LNGTH16P  DW    LNGTH16P - PARM16
LNGTH16V  DW    8

PARM17   DB    'ZCRS C'
LNGTH17P DW    LNGTH17P - PARM17
LNGTH17V DW    2

PARM47    DB    'ARBFND2 C'
LNGTH47P  DW    LNGTH47P - PARM47
LNGTH47V  DW    1

PARM51    DB    'DELAY2 I'
LNGTH51P  DW    LNGTH51P - PARM51
LNGTH51V  DW    2
```

```
PAGE
;*
;* Service Point Command Facility Variable Definitions
;*
PARM31    DB    'REQCODE3 C'
LNGTH31P  DW    LNGTH31P - PARM31
LNGTH31V  DW    1

PARM33    DB    '(PRIMRC3,PRIMEC3,PRIMET3) I'
LNGTH33P  DW    LNGTH33P - PARM33
LNGTH33V  DW    2

PARM34    DB    'TARGNAME C'
LNGTH34P  DW    LNGTH34P - PARM34
LNGTH34V  DW    8

PARM27    DB    'MSGTYPE C'
LNGTH27P  DW    LNGTH27P - PARM27
LNGTH27V  DW    1

PARM35    DB    'MSGFILE C'
LNGTH35P  DW    LNGTH35P - PARM35
LNGTH35V  DW    4

PARM36    DB    'MSGNUM C'
LNGTH36P  DW    LNGTH36P - PARM36
LNGTH36V  DW    4

PARM37    DB    'MBLENGTH I'
LNGTH37P  DW    LNGTH37P - PARM37
LNGTH37V  DW    2

PARM38    DB    'MSGCOUNT I'
LNGTH38P  DW    LNGTH38P - PARM38
LNGTH38V  DW    2

PARM39    DB    'CONVERT C'
LNGTH39P  DW    LNGTH39P - PARM39
LNGTH39V  DW    1

PAGE

PARM40    DB    '(MSGBUFRA,MSGBUFRB,MSGBUFRC,MSGBUFRD,MSGBUFRE,MSGBUFRF,MSGBUFRG,MSGBUFRH,MSGBUFRI) C'
LNGTH40P  DW    LNGTH40P - PARM40
LNGTH40V  DW    70

PARM41    DB    'CMDLGTH I'
LNGTH41P  DW    LNGTH41P - PARM41
LNGTH41V  DW    1

PARM42    DB    '(COMMANDA,COMMANDB,COMMANDC,COMMANDD) C'
LNGTH42P  DW    LNGTH42P - PARM42
LNGTH42V  DW    64

PARM43    DB    'RECVCORR C'
LNGTH43P  DW    LNGTH43P - PARM43
LNGTH43V  DW    20
```

```
PARM44    DB    'SENDCORR C'
LNGTH44P  DW    LNGTH44P - PARM44
LNGTH44V  DW    20

PARM45    DB    'FORCE C'
LNGTH45P  DW    LNGTH45P - PARM45
LNGTH45V  DW    1

PAGE

PARM48    DB    'ARBFND3 C'
LNGTH48P  DW    LNGTH48P - PARM48
LNGTH48V  DW    1

PARM52    DB    'DELAY3 I'
LNGTH52P  DW    LNGTH52P - PARM52
LNGTH52V  DW    2

PARM57    DB    'SPCFOPT I'
LNGTH57P  DW    LNGTH57P - PARM57
LNGTH57V  DW    1

PARM58    DB    'OPERNAME C'
LNGTH58P  DW    LNGTH58P - PARM58
LNGTH58V  DW    8

PARM60    DB    'NMVTNAME C'
LNGTH60P  DW    LNGTH60P - PARM60
LNGTH60V  DW    12

PARM61    DB    'SENSETYP I'
LNGTH61P  DW    LNGTH61P - PARM61
LNGTH61V  DW    1

PARM62    DB    'LCCSTAT I'
LNGTH62P  DW    LNGTH62P - PARM62
LNGTH62V  DW    1

PARM63    DB    'ERRDETAL I'
LNGTH63P  DW    LNGTH63P - PARM63
LNGTH63V  DW    1

PARM64    DB    'USERSENS C'
LNGTH64P  DW    LNGTH64P - PARM64
LNGTH64V  DW    8

PARM65    DB    'SVKEY C'
LNGTH65P  DW    LNGTH65P - PARM65
LNGTH65V  DW    2

PARM66    DB    'SFKEY C'
LNGTH66P  DW    LNGTH66P - PARM66
LNGTH66V  DW    2

PARM67    DB    'IORETCOD I'
LNGTH67P  DW    LNGTH67P - PARM67
LNGTH67V  DW    2
```

```
PARM73    DB    '(CORR1,CORR2,CORR3,CORR4,CORR5,CORR6,CORR7,CORR8) C'
LNGTH73P  DW    LNGTH73P - PARM73
LNGTH73V  DW    ASC_CORR_LENGTH

PARM74    DB    '(CORROPT,UNRSPCNT) I'
LNGTH74P  DW    LNGTH74P - PARM74
LNGTH74V  DW    1

PARM75    DB    'RECID C'
LNGTH75P  DW    LNGTH75P - PARM75
LNGTH75V  DW    2

PARM76    DB    'DISPTYPE C'
LNGTH76P  DW    LNGTH76P - PARM76
LNGTH76V  DW    1

PARM77    DB    'ARBFILE C'
LNGTH77P  DW    LNGTH77P - PARM77
LNGTH77V  DW    13

PARM78    DB    'PARSENSE C'
LNGTH78P  DW    LNGTH78P - PARM78
LNGTH78V  DW    PARSE_SENSE_ASCII_LEN

PAGE
;*
;* Host Data Transfer Facility Variable Definitions
;*
PARM19    DB    'REQCODE4 C'
LNGTH19P  DW    LNGTH19P - PARM19
LNGTH19V  DW    1

PARM21    DB    '(PRIMRC4,PRIMEC4,PRIMET4) I'
LNGTH21P  DW    LNGTH21P - PARM21
LNGTH21V  DW    2

PARM22    DB    'PCFILENM C'
LNGTH22P  DW    LNGTH22P - PARM22
LNGTH22V  DW    31

PARM23    DB    'HOSTFILE C'
LNGTH23P  DW    LNGTH23P - PARM23
LNGTH23V  DW    8

PARM24    DB    'STRTBYTE C'
LNGTH24P  DW    LNGTH24P - PARM24
LNGTH24V  DW    8

PARM25    DB    'XPC C'
LNGTH25P  DW    LNGTH25P - PARM25
LNGTH25V  DW    1

PARM26    DB    'BLKZ I'
LNGTH26P  DW    LNGTH26P - PARM26
LNGTH26V  DW    2

PARM28    DB    'NEXTBYTE C'
LNGTH28P  DW    LNGTH28P - PARM28
LNGTH28V  DW    8
```

```
PARM49    DB    'ARBFND4 C'
LNGTH49P  DW    LNGTH49P - PARM49
LNGTH49V  DW    1

PARM55    DB    'DELAY4 I'
LNGTH55P  DW    LNGTH55P - PARM55
LNGTH55V  DW    2

PARM56    DB    'XFERCOMP C'
LNGTH56P  DW    LNGTH56P - PARM56
LNGTH56V  DW    2

PAGE
;*
;* EZ-VU DISPLAY STRINGS
;*

PARM1D_KEYS     EQU     F3_OK+F10_OK+Esc_OK
PARM1D    DB    'DISPLAY DCJVAP00'
LNGTH1PD  DW    LNGTH1PD - PARM1D

PARM2D    DB    'PANDEL'
LNGTH2PD  DW    LNGTH2PD - PARM2D

PARM4D_KEYS     EQU     F3_OK+ENTER_OK+Esc_OK
PARM4D    DB    'DISPLAY DCJVMP00'
LNGTH4PD  DW    LNGTH4PD - PARM4D

PARM5D_KEYS     EQU     F3_OK+F10_OK+Esc_OK
PARM5D    DB    'DISPLAY DCJVOP00'
LNGTH5PD  DW    LNGTH5PD - PARM5D

PARM6D    DB    'SETMSG VAPI'
MSGNUM6   DB    '0000 NMVTFILE'
LNGTH6PD  DW    LNGTH6PD - PARM6D

PARM8D    DB    'DISPLAY'
LNGTH8PD  DW    LNGTH8PD - PARM8D

PARM9D    DB    'SETMSG VAPI'
MSGNUM9   DB    '0000'
LNGTH9PD  DW    LNGTH9PD - PARM9D

PARM10D   DB    'CONTROL CURSOR '
ZFLD      DB    '          '
ZCRS      DB    '   '
LNGTH10PD DW    LNGTH10PD - PARM10D

PARM11D_KEYS     EQU     F3_OK+F10_OK+Esc_OK
PARM11D   DB    'DISPLAY DCJVDP00'
LNGTH11PD DW    LNGTH11PD - PARM11D

PAGE

PARM12D    DB    'SETMSG VAPI0009 STRTBYTE'
LNGTH12PD  DW    LNGTH12PD - PARM12D
```

```
PARM13D_KEYS    EQU     F3_OK+F6_OK+F7_OK+F8_OK+F10_OK+Esc_OK
PARM13D   DB    'DISPLAY DCJVCP01'
LNGTH13PD DW    LNGTH13PD - PARM13D

PARM14D_KEYS    EQU     F3_OK+Esc_OK
PARM14D   DB    'DISPLAY DCJVCP06'
LNGTH14PD DW    LNGTH14PD - PARM14D

PARM15D    DB    'SETMSG VAPI0010'
LNGTH15PD  DW    LNGTH15PD - PARM15D

PARM16D    DB    'SETMSG VAPI0011 MSGBUFFR'
LNGTH16PD  DW    LNGTH16PD - PARM16D

PARM17D    DB    'SETMSG VAPI0012 SENDCORR'
LNGTH17PD  DW    LNGTH17PD - PARM17D

PARM18D    DB    'SETMSG VAPI0013'
LNGTH18PD  DW    LNGTH18PD - PARM18D

PARM19D_KEYS    EQU     F3_OK+F6_OK+ENTER_OK+Esc_OK
PARM19D   DB    'DISPLAY DCJVCP00'
LNGTH19PD DW    LNGTH19PD - PARM19D

PARM20D_KEYS    EQU     F3_OK+F4_OK+F6_OK+F9_OK+F10_OK+Esc_OK
PARM20D   DB    'DISPLAY DCJVCP02'
LNGTH20PD DW    LNGTH20PD - PARM20D

PARM21D_KEYS    EQU     F3_OK+F6_OK+F7_OK+F10_OK+Esc_OK
PARM21D   DB    'DISPLAY DCJVCP03'
LNGTH21PD DW    LNGTH21PD - PARM21D

PARM22D_KEYS    EQU     F3_OK+F6_OK+F8_OK+F10_OK+Esc_OK
PARM22D   DB    'DISPLAY DCJVCP04'
LNGTH22PD DW    LNGTH22PD - PARM22D

PARM23D_KEYS    EQU     F3_OK+F6_OK+F7_OK+F10_OK+Esc_OK
PARM23D   DB    'DISPLAY DCJVCP05'
LNGTH23PD DW    LNGTH23PD - PARM23D

PARM24D_KEYS    EQU     ENTER_OK+F3_OK+Esc_OK
PARM24D   DB    'DISPLAY DCJVCP07'
LNGTH24PD DW    LNGTH24PD - PARM24D

PARM25D_KEYS    EQU     F3_OK+F6_OK+F9_OK+F10_OK+Esc_OK
PARM25D   DB    'DISPLAY DCJVBP01'
LNGTH25PD DW    LNGTH25PD - PARM25D

PARM26D_KEYS    EQU     F3_OK+F6_OK+F9_OK+F10_OK+Esc_OK
PARM26D   DB    'DISPLAY DCJVBP02'
LNGTH26PD DW    LNGTH26PD - PARM26D


DATA    ENDS
```

APIMAIN.MAC

```
PAGE
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.

;*
;*  MACRO DEFINITIONS
;*

PUSHREGS MACRO
        PUSH  BP                        ;; SAVE REGISTERS
        PUSH  AX
        PUSH  BX
        PUSH  CX
        PUSH  DX
        PUSH  SI
        PUSH  DI
        PUSH  ES
        PUSH  DS
        ENDM

POPREGS  MACRO
        POP   DS                        ;; RESTORE REGISTERS
        POP   ES
        POP   DI
        POP   SI
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        POP   BP
        ENDM

;*****************************************
;* This macro is used for calls to EZVU II. *
;*****************************************

DMPC     MACRO   TYPE,PARMS
        PUSHREGS                    ;; Save all regs
        MOV     EZVU_RC,0           ;; Zero EZ-VU Return Code
        MOV     BP,SP               ;; Save stack pointer
        IRP     X,<PARMS>           ;; Push parameters onto stack
        MOV     AX,OFFSET X         ;;
        PUSH    AX                  ;;
        ENDM                        ;;
EZVU_ADDR = $                       ;;
        CALL    TYPE                ;; Call appropriate EZ-VU II rtn
        MOV     SP,BP               ;; Restore stack pointer
        POPREGS                     ;; Restore all regs
        PUSH    AX                ;; Save AX
        LEA     AX,EZVU_ADDR      ;; Store address of EZ-VU II call in
        MOV     EZVU_CALL_ADDR,AX ;; EZVU_CALL_ADDR ( parm for CHECK_EZVU_RC )
        POP     AX                ;; Restore AX
        CALL    CHECK_EZVU_RC     ;; Ensure that EZ-VU II Return code is zero
        ENDM

;***********************************************
;* This macro is used for calls to EZVU II.    *
;* and is identical to DMPC except that it does *
;* not save and restore all the registers.     *
;***********************************************

DMPC_NS MACRO   TYPE,PARMS
        MOV     EZVU_RC,0         ;; Zero EZVU Return Code
        MOV     BP,SP             ;; Save stack pointer
        IRP     X,<PARMS>         ;; Push parameters onto stack
        MOV     AX,OFFSET X       ;;
        PUSH    AX                ;;
        ENDM                      ;;
EZVU_ADDR = $                     ;;
        CALL    TYPE              ;; Call appropriate EZVU II rtn
        MOV     SP,BP             ;; Restore stack pointer
        PUSH    AX                ;; Save AX
        LEA     AX,EZVU_ADDR      ;; Store address of EZVU II call in
        MOV     EZVU_CALL_ADDR,AX ;; EZVU_CALL_ADDR ( parm for CHECK_EZVU_RC )
        POP     AX                ;; Restore AX
        CALL    CHECK_EZVU_RC     ;; Ensure that EZVU II Return code is zero
        ENDM

;***********************************************
;* This macro is used for calls to EZVU II.    *
;* and is identical to DMPC_NS except that it  *
;* does not check the return code from EZVU.   *
;***********************************************

DMPC_NC MACRO   TYPE,PARMS
        MOV     EZVU_RC,0         ;; Zero EZVU Return Code
        MOV     BP,SP             ;; Save stack pointer
        IRP     X,<PARMS>         ;; Push parameters onto stack
        MOV     AX,OFFSET X       ;;
        PUSH    AX                ;;
        ENDM                      ;;
EZVU_ADDR = $                     ;;
        CALL    TYPE              ;; Call appropriate EZVU II rtn
        MOV     SP,BP             ;; Restore stack pointer
        PUSH    AX                ;; Save AX
        LEA     AX,EZVU_ADDR      ;; Store address of EZVU II call in
        MOV     EZVU_CALL_ADDR,AX ;; EZVU_CALL_ADDR ( parm for CHECK_EZVU_RC )
        POP     AX                ;; Restore AX
;       CALL    CHECK_EZVU_RC     ;; Ensure that EZVU II Return code is zero
        ENDM


SHOWERR_MSG  MACRO  MESSAGE_NUM
             MOV AX,MESSAGE_NUM
             CALL SHOW_ERRMSG
             ENDM
```

```
MOVE_STRING  MACRO  SOURCE_STRING,TARGET_STRING,BYTE_COUNT
             PUSH ES
             PUSH SI
             PUSH DI
             PUSH CX

             PUSH DS
             POP  ES
             LEA  SI,SOURCE_STRING
             LEA  DI,TARGET_STRING
             MOV  CX,BYTE_COUNT
             CLD
REP          MOVSB

             POP CX
             POP DI
             POP SI
             POP ES
             ENDM

COMPARE_STRINGS  MACRO  SOURCE_STRING,TARGET_STRING,BYTE_COUNT
             PUSH ES
             PUSH SI
             PUSH DI
             PUSH CX

             PUSH DS
             POP  ES
             LEA  SI,SOURCE_STRING
             LEA  DI,TARGET_STRING
             MOV  CX,BYTE_COUNT
             CLD
REPE         CMPSB

             POP CX
             POP DI
             POP SI
             POP ES
             ENDM

FILL_CHAR    MACRO TARGET_AREA,FILLCHAR,BYTE_COUNT
             PUSH ES
             PUSH DI
             PUSH CX
             PUSH AX

             MOV AX,DS
             MOV ES,AX
             MOV AL,FILLCHAR
             LEA DI,TARGET_AREA
             MOV CX,BYTE_COUNT
             CLD
REP          STOSB

             POP AX
             POP CX
             POP DI
             POP ES
             ENDM
```

APIMAIN.EXR

```
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.

;*********************************************************************
;
; APIMAIN.EXR
;
; Include this file in any procedures using the variables below, which
;      are found in APIMAIN.DSG
;
;*********************************************************************


extrn CHOICE:word       ;State variable for SELMENU

extrn ZRSP1:byte              ;Scan  code of key that caused Panel Exit
extrn ZRSP2:byte              ;ASCII code of key that caused Panel Exit

extrn ZENT1:byte              ;Scan code of key to be used as Enter key
extrn ZENT2:byte              ;ASCII code of key to be used as Enter key
extrn ZENT1a:byte             ;Scan code of key to be used as Enter key
extrn ZENT2a:byte             ;Scan code of key to be used as Enter key
extrn ZENT1b:byte             ;Scan code of key to be used as Enter key
extrn ZENT2b:byte             ;Scan code of key to be used as Enter key
extrn ZENT1c:byte             ;Scan code of key to be used as Enter key
extrn ZENT2c:byte             ;Scan code of key to be used as Enter key
extrn ZENT1E:byte             ;Scan code of ESC key
extrn ZENT2E:byte             ;ASCII code of ESC key

extrn ZENT1F:byte             ;F4 key - scan  code
extrn ZENT2F:byte             ;F4 key - ASCII code

extrn ZENT1PUP:byte            ;PgUp scan code
extrn ZENT2PUP:byte            ;PgUp ASCII
extrn ZENT1PDN:byte            ;PgDn scan code
extrn ZENT2PDN:byte            ;PgDn ASCII

extrn ZENT1N:byte             ;Return key - scan  code
extrn ZENT2N:byte             ;Return key - ASCII code

extrn LNGTH9V:word
extrn ZATR:byte               ;Color used when input field is highlighted
                              ;Ebony foreground, white background

extrn IOretcod:word           ;File I/O return code for error messages.
```

APIMAIN.DEF

```
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.

;**************************************************************************
; APIMAIN.DEF
;
; Includes definitions of various constants which may be shared
;       between the main API routine and the routines which are linked
;       to it.
;**************************************************************************

;**************************************************************************
; Key Flags:  When defining keys for a panel, add up all flags of keys
;       to be made valid (set to END), and place it in the word which
;       is passed to SET_KEYS
;**************************************************************************
F1_OK           EQU     0100h           ;For F keys, if set valid, then
F2_OK           EQU     0200h           ;       key beeps if invalid
F3_OK           EQU     0400h
F4_OK           EQU     0800h
F5_OK           EQU     1000h
F6_OK           EQU     2000h
F7_OK           EQU     4000h
F8_OK           EQU     8000h
F9_OK           EQU     0001h
F10_OK          EQU     0002h
F11_OK          EQU     0004h
F12_OK          EQU     0008h

PGUP_OK         EQU     0010h           ;Beeps if invalid
PGDN_OK         EQU     0020h           ;Beeps if invalid
ESC_OK          EQU     0040h           ;Beeps if invalid

ENTER_OK        EQU     0080h           ;Acts as Tab if invalid


;**************************************************************************
; Buffer size for file I/O
;**************************************************************************
NMVTBUFF_SIZE   EQU 2048
```

APIMAIN.ASM

```
; (CTRL-OH) IBM PC PRINTER CONDENSED MODE
;       PAGE  ,132
;       TITLE API Sample Program - (C) Copyright IBM Corp. 1986,1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;  You are hereby licensed to use, reproduce, and distribute
;  these sample programs as your needs require.  IBM does not
;  warrant the suitability or integrity of these sample programs
;  and accepts no responsibility for their use for your
;  applications.  If you choose to copy and redistribute
;  significant portions of these sample programs, you should
;  preface such copies with this copyright notice.

        .SALL                           ;Suppress macro expansion

        INCLUDE  APIMAIN.DEF            ;Include shared constant definitions
        INCLUDE  APIMAIN.DSG            ;Data Segment
        INCLUDE  APIUTIL.EXR            ;Include shared procedure definitions
        IF1
          INCLUDE  APIMAIN.MAC          ;Macros
        ELSE
          %OUT Starting second pass ...
        ENDIF

PAGE

        extrn Dcjva00:far               ;Alert Manager
        extrn Dcjvo00:far               ;Operator Communications
        extrn Dcjvb00:far               ;Build and Parse routines
        extrn Dcjvc00:far               ;Service Point Command Facility
        extrn Dcjvd00:far               ;Host Data Transfer

        extrn spcf_display_init:near    ;Display variables initialization
        extrn spcf_display_pan:near     ;Display main panel routine

;*********************************************************************
; Define the utility routines and variables available in APIDISP.ASM
;*********************************************************************

extrn PJFILENC:byte                     ; file name parameter for
                                        ; spcf_display_unformatted &
                                        ; spcf_display_formatted
extrn pj_translate_fg:word              ; dump mode parameter for
                                        ; spcf_display_unformatted
extrn ebcdic_fg:abs                     ; constant for spcf_display_unformatted

extrn spcf_display_unformatted:near     ; displays hex dumps of files
extrn spcf_display_formatted:near       ; displays formatted ARB's

PGROUP  GROUP   CSEG

        PUBLIC  APITEST

CSEG    SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:PGROUP,DS:DGROUP,ES:DGROUP,SS:NOTHING
        EXTRN ISPASM:FAR               ;EZ-VU II Display functions
        EXTRN ISPASMV:FAR              ;EZ-VU II Variable definitions

;*******************************************************************
;*                                                                  *
;*  Procedure Name: APITEST                                         *
;*                                                                  *
;*  Description  : Main line procedure which runs the main menu of *
;*                 the manual tester.                               *
;*                                                                  *
;*  Input : Choice from the EZVU II panel.                          *
;*                                                                  *
;*  Output : Displays requested panel or error message if invalid  *
;*           choice was selected.                                   *
;*                                                                  *
;*******************************************************************

apitest proc    far                    ;Entry point from dos

start:
;*****************************************************************
;* The ASSUME statement shown above and the register            *
;* initialization code shown here is done for the sake of       *
;* compatibility with EZVU II. For more information see the     *
;* EZVU II Development Facility User's Guide pages 28-29        *
;*****************************************************************
       jmp reg_setup

     DB  '(C) Copyright IBM Corp. 1986,1987 ',CR,LF
     DB  'You are hereby licensed to use, reproduce, and distribute',CR,LF
     DB  'these sample programs as your needs require.  IBM does not',CR,LF
     DB  'warrant the suitability or integrity of these sample programs',CR,LF
     DB  'and accepts no responsibility for their use for your',CR,LF
     DB  'applications.  If you choose to copy and redistribute',CR,LF
     DB  'significant portions of these sample programs, you should',CR,LF
     DB  'preface such copies with this copyright notice.',CR,LF
     DB  26

reg_setup:
        push    ds
        mov     ax,dgroup
        mov     ds,ax                  ;Init Data Seg ptr
        mov     bx,ax                  ;Save DGROUP ptr in BX
        lea     ax,stktop
;*
;* Disable interrupts and swap from EZ-VU II stack to application stack
;*
        cli                            ;Disable interrupts
        mov     ss,bx                  ;Set Stack Seg ptr
        mov     sp,ax                  ;Set Stack Offset Ptr
        sti                            ;Re-enable interrupts

page


;*
;* Define all common vars and main menu vars
;*
        DMPC_NS ISPASMV,<LNGTH8P,PARM8,EZVU_RC,Zrspl,LNGTH8V>
```

```
         DMPC_NS ISPASMV,<LNGTH9P,PARM9,EZVU_RC,Zent1,LNGTH9V>
         DMPC_NS ISPASMV,<LNGTH10P,PARM10,EZVU_RC,CHOICE,LNGTH10V>
         DMPC_NS ISPASMV,<LNGTH15P,PARM15,EZVU_RC,Zatr,LNGTH15V>
         DMPC_NS ISPASMV,<LNGTH16P,PARM16,EZVU_RC,ZFLD,LNGTH16V>
         DMPC_NS ISPASMV,<LNGTH17P,PARM17,EZVU_RC,ZCRS,LNGTH17V>

         DMPC_NS ISPASMV,<LNGTH67P,PARM67,EZVU_RC,IORETCOD,LNGTH67V>



;*
;* Define vars necessary for Alert Panel
;*
         DMPC_NS ISPASMV,<LNGTH2P,PARM2,EZVU_RC,REQCODE1_ASC,LNGTH2V>
         DMPC_NS ISPASMV,<LNGTH4P,PARM4,EZVU_RC,NMVTFILE,LNGTH4V>
         DMPC_NS ISPASMV,<LNGTH5P,PARM5,EZVU_RC,NMVTTARG,LNGTH5V>
         DMPC_NS ISPASMV,<LNGTH6P,PARM6,EZVU_RC,PRIME_RC1,LNGTH6V>
         DMPC_NS ISPASMV,<LNGTH7P,PARM7,EZVU_RC,ALERT_RC1,LNGTH7V>
         DMPC_NS ISPASMV,<LNGTH46P,PARM46,EZVU_RC,ARB_FOUND1,LNGTH46V>
         DMPC_NS ISPASMV,<LNGTH50P,PARM50,EZVU_RC,DELAY1,LNGTH50V>

;*
;* Define vars necessary for Operator Communications Panel
;*
         DMPC_NS ISPASMV,<LNGTH12P,PARM12,EZVU_RC,REQCODE2_ASC,LNGTH12V>
         DMPC_NS ISPASMV,<LNGTH14P,PARM14,EZVU_RC,PRIME_RC2,LNGTH14V>
         DMPC_NS ISPASMV,<LNGTH47P,PARM47,EZVU_RC,ARB_FOUND2,LNGTH47V>
         DMPC_NS ISPASMV,<LNGTH51P,PARM51,EZVU_RC,DELAY2,LNGTH51V>

;*
;* Define vars necessary for SPCF Panel
;*
         DMPC_NS ISPASMV,<LNGTH31P,PARM31,EZVU_RC,REQCODE3_ASC,LNGTH31V>
         DMPC_NS ISPASMV,<LNGTH33P,PARM33,EZVU_RC,PRIME_RC3,LNGTH33V>
         DMPC_NS ISPASMV,<LNGTH34P,PARM34,EZVU_RC,TARGNAME,LNGTH34V>
         DMPC_NS ISPASMV,<LNGTH27P,PARM27,EZVU_RC,MSGTYPE,LNGTH27V>
         DMPC_NS ISPASMV,<LNGTH35P,PARM35,EZVU_RC,MSGFILE,LNGTH35V>
         DMPC_NS ISPASMV,<LNGTH36P,PARM36,EZVU_RC,MSGNUM,LNGTH36V>
         DMPC_NS ISPASMV,<LNGTH37P,PARM37,EZVU_RC,MBLENGTH,LNGTH37V>
         DMPC_NS ISPASMV,<LNGTH38P,PARM38,EZVU_RC,MSGCOUNT,LNGTH38V>
         DMPC_NS ISPASMV,<LNGTH39P,PARM39,EZVU_RC,CONVERT,LNGTH39V>
         DMPC_NS ISPASMV,<LNGTH40P,PARM40,EZVU_RC,MSGBUFFR1,LNGTH40V>
         DMPC_NS ISPASMV,<LNGTH41P,PARM41,EZVU_RC,CMDLGTH,LNGTH41V>
         DMPC_NS ISPASMV,<LNGTH42P,PARM42,EZVU_RC,COMMAND,LNGTH42V>
         DMPC_NS ISPASMV,<LNGTH43P,PARM43,EZVU_RC,RECVCORR_HEXASC,LNGTH43V>
         DMPC_NS ISPASMV,<LNGTH44P,PARM44,EZVU_RC,SENDCORR_HEXASC,LNGTH44V>
         DMPC_NS ISPASMV,<LNGTH45P,PARM45,EZVU_RC,FORCE,LNGTH45V>
         DMPC_NS ISPASMV,<LNGTH48P,PARM48,EZVU_RC,ARB_FOUND3,LNGTH48V>
         DMPC_NS ISPASMV,<LNGTH52P,PARM52,EZVU_RC,DELAY3,LNGTH52V>

         DMPC_NS ISPASMV,<LNGTH57P,PARM57,EZVU_RC,SPCFOPT,LNGTH57V>

         DMPC_NS ISPASMV,<LNGTH58P,PARM58,EZVU_RC,OPERNAME,LNGTH58V>

         DMPC_NS ISPASMV,<LNGTH60P,PARM60,EZVU_RC,NMVTNAME,LNGTH60V>

         DMPC_NS ISPASMV,<LNGTH61P,PARM61,EZVU_RC,SENSETYP,LNGTH61V>
         DMPC_NS ISPASMV,<LNGTH62P,PARM62,EZVU_RC,LCCSTAT,LNGTH62V>
         DMPC_NS ISPASMV,<LNGTH63P,PARM63,EZVU_RC,ERRDETAL,LNGTH63V>
         DMPC_NS ISPASMV,<LNGTH64P,PARM64,EZVU_RC,USERSENSE_ASC,LNGTH64V>
         DMPC_NS ISPASMV,<LNGTH65P,PARM65,EZVU_RC,SVKEY_ASC,LNGTH65V>
         DMPC_NS ISPASMV,<LNGTH66P,PARM66,EZVU_RC,SFKEY_ASC,LNGTH66V>

         DMPC_NS ISPASMV,<LNGTH73P,PARM73,EZVU_RC,CORR_ASC_TBL,LNGTH73V>
         DMPC_NS ISPASMV,<LNGTH74P,PARM74,EZVU_RC,CORROPT,LNGTH74V>
         DMPC_NS ISPASMV,<LNGTH75P,PARM75,EZVU_RC,RECID_ASC,LNGTH75V>
         DMPC_NS ISPASMV,<LNGTH76P,PARM76,EZVU_RC,DISPTYPE,LNGTH76V>
         DMPC_NS ISPASMV,<LNGTH77P,PARM77,EZVU_RC,ARBFILE,LNGTH77V>
         DMPC_NS ISPASMV,<LNGTH78P,PARM78,EZVU_RC,PARSE_SENSE_ASCII,LNGTH78V>

;*
;* Define vars necessary for HDF Panel
;*
         DMPC_NS ISPASMV,<LNGTH19P,PARM19,EZVU_RC,REQCODE4_ASC,LNGTH19V>
         DMPC_NS ISPASMV,<LNGTH21P,PARM21,EZVU_RC,PRIME_RC4,LNGTH21V>
         DMPC_NS ISPASMV,<LNGTH22P,PARM22,EZVU_RC,PCFILENM,LNGTH22V>
         DMPC_NS ISPASMV,<LNGTH23P,PARM23,EZVU_RC,HOSTFILENM,LNGTH23V>
         DMPC_NS ISPASMV,<LNGTH24P,PARM24,EZVU_RC,STARTBYTE_ASC,LNGTH24V>
         DMPC_NS ISPASMV,<LNGTH25P,PARM25,EZVU_RC,XPC,LNGTH25V>
         DMPC_NS ISPASMV,<LNGTH26P,PARM26,EZVU_RC,BLKZ,LNGTH26V>
         DMPC_NS ISPASMV,<LNGTH28P,PARM28,EZVU_RC,NEXTBYTE_ASC,LNGTH28V>
         DMPC_NS ISPASMV,<LNGTH49P,PARM49,EZVU_RC,ARB_FOUND4,LNGTH49V>
         DMPC_NS ISPASMV,<LNGTH55P,PARM55,EZVU_RC,DELAY4,LNGTH55V>
         DMPC_NS ISPASMV,<LNGTH56P,PARM56,EZVU_RC,XFERCOMP_ASC,LNGTH56V>


        call spcf_display_init       ;Initialize display variables for new
                                     ;        stuff


disp_main_menu:
        cmp     exitflag,1                     ;Time to quit, Y/N?
        jne     dont_exit                      ;No
        jmp     pgm_exit                       ;Yes exit program

dont_exit:
        mov     Active_Keys, PARM4D_KEYS       ;
        call    set_active_keys
        DMPC    ISPASM,<LNGTH4PD,PARM4D,EZVU_RC> ;Display main menu

        cmp     Zrspl,F3                       ;Was F3 the exit key?
        jne     not_f3_m                       ;No, check next key
        jmp     pgm_exit                       ;Yes, end the pgm.

not_f3_m:
        cmp     Zrsp2,cr                       ;Was Return the exit key?
        je      choice_alert                   ;Yes, process selection
        jmp     unknown_choice

choice_alert:
        cmp     choice,1                       ;Was it choice 1?
        jne     choice_opcomm                  ;No, check next choice
        call    alertpan                       ;Yes, run Alert Panel
        jmp     disp_main_menu                   ;Loop back to main menu

choice_opcomm:
        cmp     choice,2                       ;Was it choice 2?
        jne     choice_spcf                    ;No, check next choice
        call    opcommpan                       ;Yes, run Op Comm Panel
        jmp     disp_main_menu                  ;Loop back to main menu

page

choice_spcf:
        cmp     choice,3                        ;Was it choice 3?
        jne     choice_hdf                      ;No, check next choice
        call    spcf_men_pan                    ;Yes, run SPCF menu panel
        jmp     disp_main_menu                  ;Loop back to main menu

choice_hdf:
        cmp     choice,4                        ;Was it choice 4?
        jne     unknown_choice                  ;No, must be invalid choice
        call    hdfpan                          ;Yes run HDF Panel
        jmp     disp_main_menu                  ;Loop back to main menu

unknown_choice:
        mov  ax,1                               ;Set default choice to 1
        mov  choice,ax
        showerr_msg 8                           ;Turn on error msg
                                                ;indicating invalid choice
        jmp  disp_main_menu                     ;Loop back to main menu

;*
;* RETURN TO DOS
;*


pgm_exit:
       DMPC    ISPASM,<LNGTH2PD,PARM2D,EZVU_RC> ;Delete menu panel

ret_cd  equ     0                       ;errorlevel return code value
ret_fn  equ     4ch                     ;'return to dos' function call

        mov     ax,ret_fn*256 + ret_cd  ;return to dos function call, and
                                        ;value to be passed to errorlevel
        int     21h                     ;return to dos
                                        ;(version 2.00 or later)

apitest endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: ALERTPAN                                       *
;*                                                                 *
;*  Description  : Runs the Alert panel.                           *
;*                                                                 *
;*  Input : Variables defined for the EZVU II Alert panel NEWALERT *
;*                                                                 *
;*  Output : Return Codes, Error Classes and Error Types as well   *
;*           as the turning on of the Alert icon (AL) and error    *
;*           messages for invalid input.                           *
;*                                                                 *
;*******************************************************************
alertpan proc near

         mov     active_keys, PARM1D_KEYS       ;set active keys
        call    set_active_keys
        dmpc    ispasm,<lngth1pd,parm1d,ezvu_rc> ;display alert panel

display_alert_panel:

        cmp     Zrsp1,F10                   ;was f10 the exit key?
        je      do_alert_test               ;yes, execute the arb

        cmp     Zrsp1,F3                    ;was f3 the exit key?
        jne     not_f3_1
        jmp     exit_alert_pan              ;yes, return to
                                            ;main menu
not_f3_1:
        jmp     alert_test_done             ;invalid exit key, redisplay
page

do_alert_test:
        call do_alert                       ;perform the current
                                            ;test case
        dmpc ispasm,<lngth10pd,parm10d,ezvu_rc>;reposit cursor
alert_test_done:

        DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC> ;Redisplay panel
        jmp     display_alert_panel

exit_alert_pan:

        ret
alertpan endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: DOALERT                                        *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call for the Alert API/CS as well as the call *
;*                 and the necessary housekeeping following the    *
;*                 call to the API/CS.                             *
;*                                                                 *
;*  Input : Variables for the EZVU II Alert panel.                 *
;*                                                                 *
;*  Output : Return Codes, Error Classes and Error Types as well   *
;*           as the turning on of the Alert icon (AL) and error    *
;*           messages for invalid input.                           *
;*                                                                 *
;*******************************************************************

do_alert   proc near
           PUSHREGS                   ;Save all regs

           mov  ax,delay1             ;Delay requested amount
           call delay                 ;of time.

           mov  ax,0ffffh             ;RESET ALL RETURN CODES TO FFFF

           mov  prime_rc1,ax
           mov  prime_ec1,ax
           mov  prime_et1,ax
           mov  alert_rcl,ax
           mov  alert_ecl,ax
           mov  alert_etl,ax
           mov  cssa_rcl,ax
           mov  cssa_ecl,ax
           mov  cssa_etl,ax
           mov  host_rcl,ax
           mov  host_ecl,ax
           mov  host_etl,ax

           mov     al,reqcode1_asc        ;CONVERT ASCII CHAR INPUT BY
           lea     si,alert_rc_tbl        ;USER TO BINARY REQUEST CODE
           call    get_reqcode
           mov     req_code1,ax

           cmp  ax,0102h                  ;Is this a send request?
           je   yes_is_send               ;Yes, set up NMVT for send
           jmp  not_a_send_request        ;No, branch around NMVT setup

PAGE

yes_is_send:
           lea  di,nmvtfile               ;Point DI at the NMVT file name.
           mov  filename_ptr,di           ;Store addr in parm for READ_NMVT
           lea  di,nmvtbuff               ;Point DI at buffer into which to
                                          ;  read the Alert NMVT.
           mov  readbuff_ptr,di           ;Store addr in parm for READ_NMVT
           mov  readbuff_size,nmvtbuff_size    ;Store read buffer size parm
                                               ;for READ_NMVT.
           call read_nmvt                 ;Read the Alert NMVT from disk
           cmp  read_nmvt_stat,0          ;Was the Read successful ?
           je   alert_nmvt_read_ok        ;Yes, continue
           jmp  alert_done                ;No, Branch to subrout exit

alert_nmvt_read_ok:
not_a_send_request:

           mov     ax,ds                  ;PUT SEGMENT OF ARB IN AX
           lea     dx,arb_id1             ;PUT OFFSET  OF ARB IN DX

           mov     ax_reg,ax              ;Save AX and DX for examination
           mov     dx_reg,dx              ;by CHECK_ARB on return

           call    Dcjva00                ;Call Alert API/CS

           call check_arb                 ;Ensure that API/CS found ARB
           mov  arb_found1,al             ;Put results of CHECK_ARB
                                          ;in EZVU display variable
           jmp  alert_done                ;Branch to subrout exit

alert_done:
           POPREGS                        ;Restore all regs
           ret

do_alert   endp

PAGE
```

```
;*******************************************************************
;*                                                                 *
;*  Procedure Name: OPCOMMPAN                                      *
;*                                                                 *
;*  Description  : Runs the Operator Communications panel.         *
;*                                                                 *
;*  Input : Variables defined for the EZVU II Operator             *
;*          Communications panel NEWOPCOM.                         *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type as well       *
;*           as the turning on or clearing of the Operator         *
;*           Communications (DP) icon and error messages for       *
;*           invalid input.                                        *
;*                                                                 *
;*******************************************************************
opcommpan proc near

         mov     Active_Keys, PARM5D_KEYS
         call    set_active_keys

         DMPC    ISPASM,<LNGTH5PD,PARM5D,EZVU_RC> ;Display Op Comm panel
display_opcomm_panel:

         cmp     Zrspl,F10                   ;Was F10 the exit key?
         je      do_opcomm_test              ;Yes, Execute the ARB

         cmp     Zrspl,F3                    ;Was F3 the exit key?
         jne     not_f3_2                    ;No, check next key
         jmp     exit_opcomm_pan             ;Yes, Return to
                                             ;Main Menu
not_f3_2:
         jmp     opcomm_test_done            ;invalid exit key, redisplay

do_opcomm_test:
         call do_opcomm                      ;Perform the current
                                             ;Test Case
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor

opcomm_test_done:
         DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC> ;Redisplay screen
         jmp display_opcomm_panel

exit_opcomm_pan:

         ret
opcommpan endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_OPCOMM                                      *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call to the Operator Communications API/CS    *
;*                 well as the call and the necessary housekeeping *
;*                 following the call to the API/CS.               *
;*                                                                 *
;*  Input : Variables from the EZVU II Operator Communications      *
;*         panel.                                                   *
;*                                                                  *
;*  Output : Return Code, Error Class and Error Type as well        *
;*          as the turning on or clearing of the Operator           *
;*          Communications (DP) icon and error messages for         *
;*          invalid input.                                          *
;*                                                                  *
;*******************************************************************
do_opcomm  proc near
           PUSHREGS                      ;save all regs

           mov  ax,delay2                ;delay requested amount
           call delay                    ;of time.

           mov  ax,0ffffh                 ;Reset all return codes to ffff

           mov  prime_rc2,ax
           mov  prime_ec2,ax
           mov  prime_et2,ax

           mov     al,reqcode2_asc        ;Convert ascii char input by
           lea     si,opcomm_rc_tbl       ;User to binary request code
           call    get_reqcode
           mov     req_code2,ax

           mov     ax,ds                  ;Put segment of arb in ax
           lea     dx,arb_id2             ;Put offset  of arb in dx

           mov     ax_reg,ax              ;Save AX and DX for examination
           mov     dx_reg,dx              ;by CHECK_ARB on return

           call    Dcjvo00                ;Call the Op Comm API/CS

           call check_arb                 ;Ensure that API/CS found ARB
           mov  arb_found2,al             ;Move result to EZVU display var
           jmp  opcomm_done               ;Branch to subrout exit

opcomm_done:
           POPREGS                        ;Save all regs
           ret

do_opcomm  endp
PAGE
;*******************************************************************
;*                                                                  *
;*  Procedure Name: SPCF_RUN_PAN                                    *
;*                                                                  *
;*  Description  : Runs the Service Point Command Facility          *
;*                 RUN command panel DCJVCP01.                      *
;*                                                                  *
;*  Input : Variables defined for the EZVU II SPCF RUN command      *
;*         panel DCJVCP01.                                          *
;*                                                                  *
;*  Output : Return Code, Error Class and Error Type as well        *
;*          as the Command and Receive Correlator received from     *
;*          the host on a receive request as well as error          *
;*          messages for invalid input.                             *
;*                                                                  *
```

```
;*****************************************************************
SPCF_RUN_PAN PROC NEAR

tot_disp_spcf_run:
        mov     Active_Keys, PARM13D_KEYS       ;Set up recognized keys
        call    set_active_keys
        DMPC    ISPASM,<LNGTH13PD,PARM13D,EZVU_RC> ;Display SPCF panel

display_spcf_panel:
        cmp     Zrspl,F8                        ;Was F8 the exit key?
        je      is_msgbuff_input                ;Yes, call MSGBUFF
                                                ;panel routine.

        jmp     not_msgbuff_input               ;No, Check other keys

is_msgbuff_input:
        call    msgbuff_pan
        jmp     tot_disp_spcf_run

not_msgbuff_input:
        cmp     Zrspl,F3                        ;Was F3 the exit key?
        jne     spcf_run_was_it_f7              ;No, check next key
        jmp     exit_spcf_run_pan               ;Yes, Return to
                                                ;Main Menu
spcf_run_was_it_f7:
        cmp     Zrspl,F7                        ;Was F7 the exit key?
        jne     spcf_run_was_it_f10             ;No, check next key
        call    load_sendcorr                   ;Yes, Run the correlator
                                                ;selection menu.
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
        jmp     tot_disp_spcf_run              ;total redisplay panel

spcf_run_was_it_f10:
        cmp     Zrspl,F10                       ;Was F10 the exit key?
        je      do_spcf_run_test                ;Yes, Execute the ARB

        cmp     Zrspl,F6                        ;Was F6 the exit key?
        je      do_second_dos                   ;Yes, exit to 2nd DOS

        jmp     spcf_run_test_done              ;No, redisplay panel

do_second_dos:
                                                ;Shell out to a
        call    execpgm                         ;secondary command
                                                ;processor.

        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
        jmp     tot_disp_spcf_run              ;Total redisplay panel

do_spcf_run_test:
        cmp     byte ptr reqcode3_asc,'S'       ;Is it a send request?
        jne     no_need_to_load_msgbuff         ;No, branch around.

        cmp     byte ptr msgtype,'B'            ;Is it from a buffer?
        jne     no_need_to_load_msgbuff         ;No, branch around.

        call    load_msgbuff                    ;Load the message buffer
        cmp     loadstat,0                      ;Was load successful?
        je      no_need_to_load_msgbuff         ;Yes load was successful
        jmp     is_msgbuff_input            ;No, redisplay message input panel

no_need_to_load_msgbuff:
        call do_spcf_run                    ;Perform the current
                                            ;Test Case
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor

PAGE

spcf_run_test_done:
        DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC> ;Redisplay panel
        jmp display_spcf_panel

exit_spcf_run_pan:

            ret
spcf_run_pan endp

PAGE
;*********************************************************************
;*                                                                   *
;*  Procedure Name: MSGBUFF_PAN                                      *
;*                                                                   *
;*  Description  : Runs the Message Buffer Input panel for the       *
;*                 Service Point Command Facility NEWSPMSB           *
;*                                                                   *
;*  Input : Variables from the EZVU II Message Buffer Input          *
;*          panel.                                                   *
;*                                                                   *
;*  Output : Message(s) inputted by the user as well as              *
;*           error messages for invalid input.                       *
;*                                                                   *
;*********************************************************************
MSGBUFF_PAN PROC NEAR

;*
;* Display the message buffer input panel
;*
        mov     Active_Keys, PARM14D_KEYS
        call    set_active_keys

        DMPC    ISPASM,<LNGTH14PD,PARM14D,EZVU_RC>
display_msgbuff_panel:

        cmp     Zrspl,F3                    ;Was F3 the exit key?
        jne     not_f3_3m                   ;No, check next key
        jmp     do_loadmsg                  ;Yes,return to
                                            ;SPCF main panel.

not_f3_3m:
        jmp     not_return_2_spcf           ;No, all valid exit
                                            ;keys checked so
                                            ;Redisplay panel.
PAGE

do_loadmsg:
        jmp     exit_msgbuff_pan

not_return_2_spcf:
```

```
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>  ;Reposit cursor
         DMPC ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>    ;Redisplay panel
         jmp  display_msgbuff_panel

exit_msgbuff_pan:
         ret

msgbuff_pan endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: LOAD_MSGBUFF                                   *
;*                                                                 *
;*  Description  : Sets up a message buffer in one of the two      *
;*                 formats expected by the SPCF API/CS.            *
;*                                                                 *
;*  Input : Variables from the EZVU II Message Buffer Input        *
;*          panel and the SPCF Run Command panel and the SPCF      *
;*          Send Message Unsolicited panel.                        *
;*                                                                 *
;*  Output : Message(s) buffer ready to be sent to the SPCF        *
;*           API/CS as well as error messages for invalid input.   *
;*           On return from this routine the variable LOADSTAT will *
;*           contain zero if the load was successful and will      *
;*           contain hex FF if the load failed.                    *
;*                                                                 *
;*******************************************************************
load_msgbuff proc near
            PUSHREGS

            mov loadstat,0                  ;Init load status to good
            cmp convert,'Y'                 ;Is this one message or multi?
            je  load_multi_msg              ;Multi

;*
;* For single messages sent unconverted it is only necessary to point
;* the ARB Message Buffer Pointer at the input field for the Message
;* Buffer Input panel.
;*
            mov word ptr msgbuff_ptr, offset msgbuffrl
            jmp load_msgbuff_exit_good

load_multi_msg:
;*
;* For messages to be sent converted, it is necessary to build
;* a separate buffer from the user input buffer. In the user input buffer
;* each message is begun with a 5 character header with a format as follows:
;*   Char 1       : blank
;*   Chars 2 - 4  : 3 ASCII/Numeric chars indicating the length of the message
;*   Char 5       : blank
;* The messages must be moved one at a time from the user input buffer
;* to a new buffer where these 5 char headers will be replaced by a
;* one byte binary field.
;*

            mov ax,ds                       ;Set ES = DS as both
            mov es,ax                       ;buffers are in the DATA segment
            lea si,msgbuffrl                ;Put offset of User input
```

```
                                          ;message buffer in SI
             lea di,msgbuffr2             ;Put offset of build
                                          ;buffer in DI
PAGE

             mov word ptr msgbuff_ptr, di ;Point message buffer pointer
                                          ;in ARB at the build buffer
             mov currmsg_num,0            ;Init current message number to zero

msg_load_loop:
             mov  ax,currmsg_num
             cmp  ax,msgcount             ;ARE WE THRU, Y/N?
             jne  load_next_msg           ;NO, DO THE NEXT MSG
             jmp  load_msgbuff_exit_good  ;YES ,EXIT RTN

load_next_msg:
             inc  si                      ;BUMP PAST LEADING BLANK
             call decasc2bin              ;CONVERT LENGTH CHARS
                                          ;TO BINARY
             cmp  cx,-1                   ;CONVERTED OK, Y/N?
             je   load_msgbuff_exit_bad   ;NO, EXIT RTN

             cmp  cx,0                    ;Is msg length = 0?
             je   load_msgbuff_exit_bad   ;Yes, exit rtn

             cmp  cx,255                  ;Is msg length > 255?
             ja   load_msgbuff_exit_bad   ;Yes, exit rtn

             mov  [di],cl                 ;YES, LOAD MSG
             inc  di                      ;BUMP PAST LENGTH BYTE
             add  si,4                    ;BUMP PAST LENGTH CHARS
                                          ;AND TRAILING BLANK

             cld                          ;ENABLE AUTO-INCREMENT
         rep movsb                        ;MOVE THE MESSAGE
             inc  currmsg_num             ;INC CURRENT MSG NUM
             jmp  msg_load_loop           ;Process next message

load_msgbuff_exit_bad:
             mov  loadstat,0ffh           ;Indicate unsuccessful
                                          ;conversion.
load_msgbuff_exit_good:

             POPREGS                      ;Restore all regs
             ret
load_msgbuff endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_SPCF_RUN                                    *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call to the Service Point Command Facility    *
;*                 API/CS as well as the call and the necessary    *
;*                 housekeeping following the call to the API/CS.  *
;*                                                                 *
;*  Input : Variables from the EZVU II SPCF Run Command panel.     *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type as well        *
;*           as the Command and Receive Correlator received from    *
;*           the host on a receive request as well as error         *
;*           messages for invalid input.                            *
;*                                                                  *
;*******************************************************************
do_spcf_run  proc near
           PUSHREGS                          ;Save all regs

           mov  ax,delay3                    ;Delay requested amount
           call delay                        ;of time.

           mov  ax,0ffffh                    ;RESET ALL RETURN CODES TO FFFF

           mov  prime_rc3,ax
           mov  prime_ec3,ax
           mov  prime_et3,ax


           mov      cmdlgth,0            ;Zero command length for
                                         ;display purposes.

           mov      al,reqcode3_asc      ;CONVERT ASCII CHAR INPUT BY
           lea      si,spcf_rc_tbl       ;USER TO BINARY REQUEST CODE
           call     get_reqcode
           mov      req_code3,ax

           cmp      reqcode3_asc,'S'     ;Is it a send request ?
           je       send_req
           jmp      not_send_req
PAGE

SEND_REQ:
;*
;* Convert the Send Correlator input by the user from the 20 Hex/ASCII
;* digits input to 10 Hex/Binary digits in the appropriate slot in
;* the ARB.
;*
           call cnv_sendcorr            ;Do the conversion.
           cmp  sendcorr_stat,0         ;Was conversion successful?
           je   sendcorr_cnv_good_run   ;Yes
           jmp  spcf_run_done_exit      ;No, exit routine

sendcorr_cnv_good_run:
not_send_req:
           mov      ax,ds                ;put segment of arb in ax
           lea      dx,arb_id3           ;put offset  of arb in dx

           mov      ax_reg,ax            ;Save AX and DX for examination
           mov      dx_reg,dx            ;by CHECK_ARB on return

           call     Dcjvc00              ;CALL THE SPCF API/CS

           call check_arb                ;Ensure that API/CS found ARB
           mov  arb_found3,al            ;Move result to EZVU display var
           jmp  spcf_done_good           ;Process results of call

spcf_done_good:
           FILL_CHAR command,' ',256      ;clear the command display buffer
```

```
                                        ;clear recvcorr display buffer
          FILL_CHAR recvcorr_hexasc,' ',asc_corr_length

          cmp  reqcode3_asc,'S'         ;if request was send
          je   del_send_run             ;command then

          cmp  prime_rc3,0              ;Was call successful ?
          je   spcf_run_goodrc          ;Yes
          jmp  spcf_run_done_exit       ;No, exit subrout

del_send_run:
          call del_sendcorr           ;Delete send correlator from
                                      ;table of outstanding correlators.
          jmp  spcf_run_done_exit     ;Exit subrout

spcf_run_goodrc:
          cmp  reqcode3_asc,'R'       ;IF REQUEST WAS RECEIVE
          je   load_command           ;COMMAND THEN

          cmp  reqcode3_asc,'C'       ;IF REQUEST WAS CLOSE
          je   del_all_corr_run       ;COMMAND THEN

          jmp  spcf_run_done_exit

del_all_corr_run:                     ;Good close
          call clear_corr_tbl         ;Clear correlator table
          jmp  spcf_run_done_exit     ;and zero count.


PAGE

load_command:
          xor  cx,cx                  ;load the command display
          mov  cl,cmdlgth             ;buffer from the real
          mov  ax,ds                  ;command buffer pointed at
          mov  es,ax                  ;by command_ptr
          lea  di,command
          lds  si,command_ptr
          cld
      rep movsb
          mov  ax,es                  ;Restore DS
          mov  ds,ax

          call cnv_recvcorr           ;Convert the Receive correlator
                                      ;to Hex/ASCII form so that it
                                      ;can be displayed.

          call save_recvcorr          ;Save Receive correlator in
                                      ;outstanding correlator table.
          jmp  spcf_run_done_exit



spcf_run_done_exit:
          POPREGS                     ;Restore all regs
          ret

do_spcf_run  endp
```

```
PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: SPCF_GNP_PAN                                   *
;*                                                                 *
;*  Description  : Runs the Service Point Command Facility         *
;*                 Get No Parse panel DCJVCP02.                    *
;*                                                                 *
;*  Input : Variables defined for the EZVU II SPCF GNP             *
;*          panel DCJVCP02.                                        *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type and Receive   *
;*           Correlator as well as the data parsed from the NMVT   *
;*           received. The parsed data includes the target         *
;*           application name, the Major vector key and length,    *
;*           the command received and command length, the list     *
;*           of link segment names and their lengths and the       *
;*           test count. Which of these data items is parsed       *
;*           depends on the key of the major vector received.      *
;*                                                                 *
;*******************************************************************
spcf_gnp_pan proc near

tot_disp_gnp:
        mov     Active_Keys, PARM20D_KEYS
        call    set_active_keys
        DMPC    ISPASM,<LNGTH20PD,PARM20D,EZVU_RC> ;Display SPCF GNP panel

display_spgnp_panel:
        cmp     Zrspl,F4                      ;Was F4  the exit key?
        jne     not_f4_spgnp                  ;No, check next key
        jmp     parse_disp_spcf_gnp           ;Yes, parse & display the NMVT

not_f4_spgnp:
        cmp     Zrspl,F9                      ;Was F9  the exit key?
        jne     not_f9_spgnp                  ;No, check next key
        jmp     hex_disp_spcf_gnp             ;Yes, Display the NMVT in hex

not_f9_spgnp:
        cmp     Zrspl,F10                     ;Was F10 the exit key?
        jne     not_f10_spgnp                 ;No, check next key
        jmp     do_spcf_gnp_test              ;Yes, Execute the ARB

not_f10_spgnp:
        cmp     Zrspl,F3                      ;Was F3 the exit key?
        jne     not_f3_spgnp                  ;No, check next key
        jmp     exit_spcf_gnp_pan             ;Yes, Return to
                                              ;SPCF Menu
not_f3_spgnp:
        cmp     Zrspl,F6                      ;Was F6 the exit key?
        je      do_2nd_dos_gnp                ;Yes
        jmp     spgnp_test_done               ;No, redisplay
                                              ;the panel.
do_2nd_dos_gnp:                               ;Shell out to a
        call    execpgm                       ;secondary command
                                              ;processor.
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
        jmp     tot_disp_gnp                  ;Total redisplay panel

do_spcf_gnp_test:
        call do_spcf_gnp                      ;Perform the current
                                              ;Test Case
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit Cursor

spgnp_test_done:
        DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC> ;Redisplay panel
        jmp     display_spgnp_panel

hex_disp_spcf_gnp:
        cmp     byte ptr nmvtname,' '       ;Is file name blank ?
        jne     cont_hex_disp               ;no, continue
        SHOWERR_MSG 300                     ;yes, show error msg and
        jmp     spgnp_test_done             ;     redisplay panel

cont_hex_disp:
        MOVE_STRING NMVTNAME,PJFILENC,12    ;Set up input file name for display
        mov     pj_translate_fg,ebcdic_fg   ;indicate EBCDIC dump
        call    spcf_display_unformatted    ;display the hex dump
        jmp     tot_disp_gnp                ;Total redisplay panel

parse_disp_spcf_gnp:
        cmp     byte ptr nmvtname,' '       ;Is file name blank ?
        jne     cont_parse_displ            ;no, continue
        SHOWERR_MSG 300                     ;yes, show error msg and
        jmp     spgnp_test_done             ;     redisplay panel

cont_parse_displ:
        MOVE_STRING NMVTname,NMVTfile,12    ;set up input  file for do_parse
        MOVE_STRING ARB_temp,ARBfile,12     ;set up output file for do_parse
        call    do_parse                    ;parse the NMVT to a file

        cmp     do_parse_rc,0               ;Was file parsed successfully ?
        je      cont_parse_disp2            ;yes, continue
        jmp     spgnp_test_done             ;no,  redisplay panel

cont_parse_disp2:
        MOVE_STRING ARB_temp,PJFILENC,12    ;set up input file for display
        call    spcf_display_formatted      ;display the formatted ARB
        jmp     tot_disp_gnp                ;Total redisplay panel

exit_spcf_gnp_pan:
        ret
spcf_gnp_pan endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_SPCF_GNP                                    *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call to the Service Point Command Facility    *
;*                 API/CS as well as the call and the necessary    *
;*                 housekeeping following the call to the API/CS.  *
;*                                                                 *
;*  Input : Variables from the EZVU II SPCF Get No Parse panel.    *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type and Receive     *
;*           Correlator as well as the data parsed from the NMVT     *
;*           received. The parsed data includes the target           *
;*           application name, the Major vector key and length,      *
;*           the command received and command length, the list       *
;*           of link segment names and their lengths and the         *
;*           test count. Which of these data items is parsed         *
;*           depends on the key of the major vector received.        *
;*                                                                    *
;*********************************************************************
do_spcf_gnp  proc near
           PUSHREGS                          ;Save all regs

           mov  ax,delay3                    ;Delay requested amount
           call delay                        ;of time.

           mov  ax,0ffffh                    ;reset all return codes to ffff

           mov  prime_rc3,ax
           mov  prime_ec3,ax
           mov  prime_et3,ax

           mov     al,reqcode3_asc           ;convert ascii char input by
           lea     si,spcf_rc_tbl            ;user to binary request code
           call    get_reqcode
           mov     req_code3,ax

PAGE

           mov     ax,ds                 ;put segment of arb in ax
           lea     dx,arb_id3            ;put offset  of arb in dx

           mov     ax_reg,ax             ;save ax and dx for examination
           mov     dx_reg,dx             ;by check_arb on return
           call    Dcjvc00               ;call the spcf api/cs

           call check_arb                ;ensure that api/cs found arb
           mov  arb_found3,al            ;move result to ezvu display var

           mov  cmdlgth,0                 ;clear command length field
                                          ;clear recvcorr display buffer
           FILL_CHAR recvcorr_hexasc,' ',asc_corr_length

           mov  word ptr recid_asc,'  '   ;clear record id

           cmp  prime_rc3,0               ;was request successful ?
           je   gnp_goodrc                ;yes
           jmp  spcf_gnp_done_exit        ;no, exit subrout

gnp_goodrc:
           cmp  reqcode3_asc,'G'          ;if request was get no parse
           je   got_no_parse              ;command then

           cmp  reqcode3_asc,'C'          ;if request was close
           je   del_all_corr_gnp          ;command then

           jmp  spcf_gnp_done_exit       ;Exit subrout

del_all_corr_gnp:                        ;Good close
```

```
          call clear_corr_tbl          ;Clear correlator table
          jmp  spcf_gnp_done_exit      ;and zero count.

got_no_parse:
          call cnv_recvcorr            ;Convert the Receive correlator
                                       ;to Hex/ASCII form so that it
                                       ;can be displayed.
          call save_recvcorr           ;Save Receive correlator in table

          mov  al,recid
          lea  di,recid_asc            ;Convert Record ID from ARB to
          call hexb2asc                ;  ASCII for display

          mov  filename_ptr, offset NMVTname
          les  di, command_ptr                ;Get NMVT address in ES:DI
          mov  word ptr Writebuff_Ptr_Tbl,di   ;Save the offset
          mov  word ptr Writebuff_Ptr_Tbl+2,es ;And the segment

          mov  ah,byte ptr es:[di]              ;Get NMVT length
          mov  al,byte ptr es:[di+1]            ;from 1st 2 bytes of NMVT.
          mov  word ptr Writebuff_Ptr_Tbl+4,ax  ;Set up NMVT length

          lea  bx,recvcorr                      ; JOF 6-2-87
          mov  word ptr Writebuff_Ptr_Tbl+6,bx  ; Save offset  of correlator
          mov  word ptr Writebuff_Ptr_Tbl+8,ds  ; Save segment of correlator
          mov  word ptr Writebuff_Ptr_Tbl+10,10 ; Save length  of correlator

          mov  word ptr Writebuff_Ptr_Tbl+16,0  ; Mark end of
                                                ; Writebuff_Ptr_Tbl

          Call Write_File             ;All OK, so save NMVT

spcf_gnp_done_exit:
          POPREGS                      ;Restore all regs
          ret

do_spcf_gnp  endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: SPCF_PUF_PAN                                   *
;*                                                                 *
;*  Description  : Runs the Service Point Command Facility         *
;*                 Put Unformatted panel DCJVCP03.                 *
;*                                                                 *
;*  Input : Variables defined for the EZVU II SPCF PUF             *
;*          panel DCJVCP02.                                        *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type.              *
;*                                                                 *
;*******************************************************************
spcf_puf_pan proc near

tot_disp_puf:
        mov     Active_Keys, PARM21D_KEYS
        call    set_active_keys
        DMPC    ISPASM,<LNGTH21PD,PARM21D,EZVU_RC> ;Display SPCF PUF panel
display_sppuf_panel:

        cmp     Zrspl,F10                    ;Was F10 the exit key?
        jne     not_f10_sppuf                ;No, check next key
        jmp     do_spcf_puf_test             ;Yes, Execute the ARB

not_f10_sppuf:
        cmp     Zrspl,F3                     ;Was F3 the exit key?
        jne     not_f3_sppuf                 ;No, check next key
        jmp     exit_spcf_puf_pan            ;Yes, Return to
                                             ;SPCF Menu
page

not_f3_sppuf:
        cmp     Zrspl,F6                     ;Was F6 the exit key?
        jne     not_f6_sppuf                 ;No, check next key
                                             ;Yes, Shell out to a
        call    execpgm                      ;secondary command
                                             ;processor.
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
        jmp     tot_disp_puf                 ;Total redisplay panel

not_f6_sppuf:
        cmp     Zrspl,F7                     ;Was F7 the exit key?
        je      corr_menu_puf                ;Yes
        jmp     sppuf_test_done              ;No, redisplay
                                             ;the panel.
corr_menu_puf:
        call    load_sendcorr                ;Run the correlator
                                             ; selection menu.
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;
        jmp     tot_disp_puf                 ;Total redisplay panel

do_spcf_puf_test:
        call do_spcf_puf                     ;Perform the current
                                             ;Test Case
        dmpc ispasm,<lngth10pd,parm10d,ezvu_rc>;Reposit Cursor

sppuf_test_done:
        dmpc    ispasm,<lngth8pd,parm8d,ezvu_rc> ;Redisplay panel
        jmp     display_sppuf_panel

exit_spcf_puf_pan:

        ret
spcf_puf_pan endp

PAGE
;*********************************************************************
;*                                                                   *
;*  Procedure Name: DO_SPCF_PUF                                      *
;*                                                                   *
;*  Description   : Performs all the preparation for execution of    *
;*                  a call to the Service Point Command Facility     *
;*                  API/CS as well as the call and the necessary     *
;*                  housekeeping following the call to the API/CS.   *
;*                                                                   *
;*  Input : Variables from the EZVU II SPCF Put Unformatted panel.   *
;*                                                                   *
;*  Output : Return Code, Error Class and Error Type.               *
;*                                                                   *
;*********************************************************************
do_spcf_puf   proc near
          PUSHREGS                      ;save all regs

          mov  ax,delay3                ;delay requested amount
          call delay                    ;of time.

          mov  ax,0ffffh                ;reset all return codes to ffff

          mov  prime_rc3,ax
          mov  prime_ec3,ax
          mov  prime_et3,ax

          mov     al,reqcode3_asc        ;convert ascii char input by
          lea     si,spcf_rc_tbl         ;user to binary request code
          call    get_reqcode
          mov     req_code3,ax

          cmp  reqcode3_asc,'P'          ;Is this a Put unformatted request?
          je   yes_is_puf                ;Yes, set up NMVT for Put
          jmp  not_a_puf_request         ;No, branch around NMVT setup

PAGE

YES_IS_PUF:
;*
;* Convert the Send Correlator input by the user from the 20 Hex/ASCII
;* digits input to 10 Hex/Binary digits in the appropriate slot in
;* the ARB.
;*
          call cnv_sendcorr             ;Do the conversion.
          cmp  sendcorr_stat,0          ;Was conversion successful?
          je   sendcorr_cnv_good_puf    ;Yes
          jmp  spcf_puf_done            ;No, exit routine

sendcorr_cnv_good_puf:
          lea  di,nmvtname               ;Point DI at the NMVT file name.
          mov  filename_ptr,di           ;Store addr
          lea  di,putreply               ;Point DI at buffer into which to
                                         ;  read the SPCF NMVT.
          mov  word ptr putrply_ptr,di
          mov  word ptr putrply_ptr+2,ds
          mov  readbuff_ptr,di           ;Store addr

 ;Store read buffer size parm for READ_NMVT.
          mov  readbuff_size,putreply_buff_size

          call read_nmvt                 ;Read the SPCF NMVT from disk
          mov  ax,filesize               ;Get length of NMVT which
                                         ;was returned by READ_NMVT

          sub  ax,10                     ;Subtract length of correlator
                                         ;which was tacked on to the end
                                         ;of the file.

          mov  putrply_len,ax            ;Store the NMVT length in
```

```
                                          ;the PUTRPLY_LEN of ARB

          cmp  read_nmvt_stat,0           ;Was the Read successful ?
          je   spcf_puf_nmvt_read_ok      ;Yes, continue
          jmp  spcf_puf_done              ;No, Branch to subrout exit

spcf_puf_nmvt_read_ok:
not_a_puf_request:

          mov     ax,ds                   ;PUT SEGMENT OF ARB IN AX
          lea     dx,arb_id3              ;PUT OFFSET  OF ARB IN DX

          mov     ax_reg,ax               ;Save AX and DX for examination
          mov     dx_reg,dx               ;by CHECK_ARB on return

          call    Dcjvc00                 ;Call SPCF API/CS

          call check_arb                  ;Ensure that API/CS found ARB
          mov  arb_found3,al              ;Put results of CHECK_ARB
                                          ;in EZVU display variable

          cmp     reqcode3_asc,'P'        ;Is this a Put unformatted request?
          jne     spcf_puf_not_put
          call    del_sendcorr            ;Delete send correlator from
                                          ;table of outstanding correlators.
          jmp     spcf_puf_done           ;Branch to subrout exit

spcf_puf_not_put:
          cmp     prime_rc3,0             ;Was request successful ?
          jne     spcf_puf_done           ;No

          cmp     reqcode3_asc,'C'        ;Is this a Close request?
          jne     spcf_puf_done           ;No, exit subrout
          call    clear_corr_tbl          ;Yes clear the correlator table
                                          ;and count.
spcf_puf_done:
          POPREGS                         ;Restore all regs
          ret

do_spcf_puf   endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: SPCF_SUN_PAN                                   *
;*                                                                 *
;*  Description  : Runs the SPCF Send Message Unsolicited panel.   *
;*                                                                 *
;*  Input : Variables defined for the EZVU II NEWSPSUN panel       *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type.              *
;*                                                                 *
;*******************************************************************
spcf_sun_pan proc near

tot_disp_spcf_sun:
        mov     Active_Keys, PARM22D_KEYS
        call    set_active_keys
```

```
         DMPC    ISPASM,<LNGTH22PD,PARM22D,EZVU_RC> ;Display SPCF panel
display_spcf_sun_panel:

         cmp     Zrspl,F8                    ;Was F8 the exit key?
         je      is_msgbuff_input_sun        ;Yes, call MSGBUFF
                                             ;panel routine.

         jmp     not_msgbuff_input_sun       ;No, Check other keys

is_msgbuff_input_sun:
         call    msgbuff_pan
         jmp     tot_disp_spcf_sun           ;display main
                                             ;SPCF panel.
not_msgbuff_input_sun:
         cmp     Zrspl,F3                    ;Was F3 the exit key?
         jne     spcf_sun_was_it_f10         ;No, check next key
         jmp     exit_spcf_sun_pan           ;Yes, Return to
                                             ;Main Menu
spcf_sun_was_it_f10:
         cmp     Zrspl,F10                   ;Was F10 the exit key?
         je      do_spcf_sun_test            ;Yes, Execute the ARB

         cmp     Zrspl,F6                    ;Was F6 the exit key?
         je      do_second_dos_sun           ;Yes, exit to 2nd DOS

         jmp     spcf_sun_test_done          ;No, redisplay panel

do_second_dos_sun:
                                             ;Shell out to a
         call    execpgm                     ;secondary command
                                             ;processor.

         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
         jmp     tot_disp_spcf_sun          ;total redisplay panel

do_spcf_sun_test:
         cmp     byte ptr reqcode3_asc,'M'   ;Is it a send msg request?
         jne     no_need_to_load_msgbuff_sun ;No, branch around.

         mov     convert,'Y'
         cmp     byte ptr msgtype,'B'        ;Is it from a buffer?
         jne     no_need_to_load_msgbuff_sun ;No, branch around.

         call    load_msgbuff                ;Load the message buffer
         cmp     loadstat,0                  ;Was load successful?
         je      no_need_to_load_msgbuff_sun ;Yes load was successful
         jmp     is_msgbuff_input_sun            ;No, redisplay message input panel

no_need_to_load_msgbuff_sun:
         call do_spcf_sun                    ;Perform the current
                                             ;Test Case
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor

PAGE

spcf_sun_test_done:
         DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC> ;Redisplay panel
         jmp display_spcf_sun_panel
```

```
exit_spcf_sun_pan:

             ret
spcf_sun_pan endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_SPCF_SUN                                    *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call to the Service Point Command Facility    *
;*                 API/CS as well as the call and the necessary    *
;*                 housekeeping following the call to the API/CS.  *
;*                                                                 *
;*  Input : Variables from the EZVU II Service Point Command       *
;*          for the panel NEWSPSUN.                                *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type.              *
;*                                                                 *
;*******************************************************************
do_spcf_sun  proc near
           PUSHREGS                   ;Save all regs

           mov  ax,delay3             ;Delay requested amount
           call delay                 ;of time.

           mov  ax,0ffffh             ;Reset all return codes to ffff

           mov  prime_rc3,ax
           mov  prime_ec3,ax
           mov  prime_et3,ax

           mov     al,reqcode3_asc    ;Convert ascii char input by
           lea     si,spcf_rc_tbl     ;User to binary request code
           call    get_reqcode
           mov     req_code3,ax

PAGE

           mov     ax,ds              ;Put segment of arb in ax
           lea     dx,arb_id3         ;Put offset  of arb in dx

           mov     ax_reg,ax          ;Save AX and DX for examination
           mov     dx_reg,dx          ;by CHECK_ARB on return
           call    Dcjvc00            ;CALL THE SPCF API/CS

           call check_arb             ;Ensure that API/CS found ARB
           mov  arb_found3,al         ;Move result to EZVU display var

           cmp  prime_rc3,0           ;If it was a successful close
           jne  do_spcf_sun_exit      ;request then clear the
           cmp  reqcode3_asc,'C'      ;correlator table and
           jne  do_spcf_sun_exit      ;zero the outstanding correlator
           call clear_corr_tbl        ;count.

do_spcf_sun_exit:
           POPREGS                    ;Restore all regs
           ret
```

```
do_spcf_sun  endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: SPCF_SER_PAN                                   *
;*                                                                 *
;*  Description  : Runs the SPCF Send Error panel.                 *
;*                                                                 *
;*  Input : Variables defined for the EZVU II NEWSPSER panel       *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type.              *
;*                                                                 *
;*******************************************************************
spcf_ser_pan proc near

tot_disp_ser:
         mov     Active_Keys, PARM23D_KEYS
         call    set_active_keys

         DMPC    ISPASM,<LNGTH23PD,PARM23D,EZVU_RC> ;Display SPCF SER panel

display_spser_panel:
         cmp     Zrspl,F10                      ;Was F10 the exit key?
         jne     not_f10_spser                  ;No, check next key
         jmp     do_spcf_ser_test               ;Yes, Execute the ARB

not_f10_spser:
         cmp     Zrspl,F3                       ;Was F3 the exit key?
         jne     not_f3_spser                   ;No, check next key
         jmp     exit_spcf_ser_pan              ;Yes, Return to
                                                ;SPCF Menu
PAGE

not_f3_spser:
         cmp     Zrspl,F6                       ;Was F6 the exit key?
         jne     not_f6_spser                   ;No, check next key
                                                ;Yes, Shell out to a
         call    execpgm                        ;secondary command
                                                ;processor.
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
         jmp     tot_disp_ser                   ;Total redisplay panel

not_f6_spser:
         cmp     Zrspl,F7                       ;Was F7 the exit key?
         je      corr_menu_ser                  ;Yes
         jmp     spser_test_done                ;No, redisplay
                                                ;the panel.
corr_menu_ser:
         call    load_sendcorr                  ;Yes, Run the correlator
                                                ;selection menu.
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit cursor
         jmp     tot_disp_ser                   ;Total redisplay panel

do_spcf_ser_test:
         call do_spcf_ser                       ;Perform the current
                                                ;Test Case
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>;Reposit Cursor

spser_test_done:
         DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC> ;Redisplay panel
         jmp     display_spser_panel

exit_spcf_ser_pan:

         ret
spcf_ser_pan endp

PAGE

;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_SPCF_SER                                    *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of  *
;*                 a call to the Service Point Command Facility    *
;*                 API/CS as well as the call and the necessary    *
;*                 housekeeping following the call to the API/CS.  *
;*                                                                 *
;*  Input : Variables from the EZVU II Service Point Command       *
;*          Facility SER Panel.                                    *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type.              *
;*                                                                 *
;*******************************************************************
do_spcf_ser  proc near
           PUSHREGS                          ;Save all regs

           mov  ax,delay3                    ;Delay requested amount
           call delay                        ;of time.

           mov  ax,0ffffh                    ;reset all return codes to ffff

           mov  prime_rc3,ax
           mov  prime_ec3,ax
           mov  prime_et3,ax

           mov     al,reqcode3_asc       ;Convert ascii char input by
           lea     si,spcf_rc_tbl        ;User to binary request code
           call    get_reqcode
           mov     req_code3,ax

           cmp     reqcode3_asc,'E'      ;Is it a Send Error request ?
           je      ser_request           ;Yes
           jmp     not_ser_request       ;No, branch around conversions

PAGE

SER_REQUEST:
;*
;* Convert the Send Correlator input by the user from the 20 Hex/ASCII
;* digits input to 10 Hex/Binary digits in the appropriate slot in
;* the ARB.
;*
           call cnv_sendcorr           ;Do the conversion.
           cmp  sendcorr_stat,0        ;Was conversion successful?
```

```
        je   sendcorr_cnv_good_ser    ;Yes
        jmp  spcf_ser_done_exit       ;No, exit routine

sendcorr_cnv_good_ser:
        mov  cx,2                     ;Put length of SVKEY_ASC in CX
        lea  di,svkey_asc             ;Point DI at SVKEY_ASC
        call asc2hex                  ;Convert ASCII to binary
                                      ;Value is returned in AX.
        cmp  cx,-1                    ;Was conversion successful ?
        jne  svkey_converted_good     ;Yes

        showerr_msg 27                ;Show error msg   nonhex chars in fld
        jmp  spcf_ser_done_exit       ;No, exit routine


svkey_converted_good:
        mov  svkey,al                 ;Store binary subvector key in ARB

        mov  cx,2                     ;Put length of SFKEY_ASC in CX
        lea  di,sfkey_asc             ;Point DI at SFKEY_ASC
        call asc2hex                  ;Convert ASCII to binary
                                      ;Value is returned in AX.
        cmp  cx,-1                    ;Was conversion successful ?
        jne  sfkey_converted_good     ;Yes

        showerr_msg 28                ;Show error msg   nonhex chars in fld
        jmp  spcf_ser_done_exit       ;No, exit routine


sfkey_converted_good:
        mov  sfkey,al                 ;Store binary subfield key in ARB

        mov  cx,4                     ;Put length of first 4 bytes of
                                      ;USERSENSE_ASC in CX.
        lea  di,usersense_asc         ;Point DI at first 4 bytes of
                                      ;USERSENSE_ASC.
        call asc2hex                  ;Convert ASCII to BINARY
        cmp  cx,-1                    ;Was conversion successful ?
        jne  user_cnv_good1           ;Yes

        showerr_msg 29                ;Show error msg   nonhex chars in fld
        jmp  spcf_ser_done_exit       ;No, exit routine

user_cnv_good1:
        mov  byte ptr usersense  ,ah  ;Store binary user sense data in ARB
        mov  byte ptr usersense+1,al

        mov  cx,4                     ;Put length of second 4 bytes of
                                      ;USERSENSE_ASC in CX.
        lea  di,usersense_asc+4       ;Point DI at second 4 bytes of
                                      ;USERSENSE_ASC.
        call asc2hex                  ;Convert ASCII to BINARY

        cmp  cx,-1                    ;Was conversion successful ?
        jne  user_cnv_good2           ;Yes

        showerr_msg 29                ;Show error msg   nonhex chars in fld
        jmp  spcf_ser_done_exit       ;No, exit routine
```

```
user_cnv_good2:
          mov  byte ptr usersense+2,ah  ;Store binary user sense data in ARB
          mov  byte ptr usersense+3,al

not_ser_request:
          mov     ax,ds                 ;put segment of arb in ax
          lea     dx,arb_id3            ;put offset  of arb in dx

          mov     ax_reg,ax             ;Save AX and DX for examination
          mov     dx_reg,dx             ;by CHECK_ARB on return
          call    Dcjvc00               ;CALL THE SPCF API/CS

          call check_arb                ;Ensure that API/CS found ARB
          mov  arb_found3,al            ;Move result to EZVU display var

          cmp  reqcode3_asc,'E'         ;Is it a Send Error request ?
          je   del_send_ser             ;Yes

          cmp  prime_rc3,0              ;Was call successful ?
          je   spcf_ser_goodrc          ;Yes
          jmp  spcf_ser_done_exit       ;No, exit subrout

spcf_ser_goodrc:
          cmp  reqcode3_asc,'C'         ;Is it a Close request ?
          jne  spcf_ser_done_exit       ;No
          call clear_corr_tbl           ;Yes, clear the correlator table.
          jmp  spcf_ser_done_exit

del_send_ser:
          call del_sendcorr             ;Delete send correlator from
                                        ;table of outstanding correlators.
          jmp  spcf_ser_done_exit

spcf_ser_done_exit:
          POPREGS                       ;Restore all regs
          ret

do_spcf_ser  endp


PAGE
;*****************************************************************
;*                                                               *
;*  Procedure Name: HDFPAN                                       *
;*                                                               *
;*  Description  : Runs the Host Data Facility panel.            *
;*                                                               *
;*  Input : Variables defined below from the EZVU II HDF panel   *
;*                                                               *
;*  Output : Return Code, Error Class and Error Type as well     *
;*           as the Offset into the file and the completion byte *
;*           on a status request as well as error messages for   *
;*           invalid input.                                      *
;*                                                               *
;*****************************************************************
hdfpan    proc near

        mov     Active_Keys, PARM11D_KEYS
        call    set_active_keys
        DMPC    ISPASM,<LNGTH11PD,PARM11D,EZVU_RC>
display_hdf_panel:

        cmp     Zrsp1,F10                   ;Was F10 the exit key?
        je      do_hdf_test                 ;Yes, Execute the ARB

        cmp     Zrsp1,F3                    ;Was F3 the exit key?
        jne     not_f3_4
        jmp     exit_hdf_pan                ;Yes, Return to
                                            ;Main Menu
PAGE

not_f3_4:
        cmp     Zrsp2,F3                    ;Was F3 the exit key?
        jne     hdf_test_done               ;No, redisplay
                                            ;the panel.
        jmp     exit_hdf_pan                ;Yes, Return to
                                            ;Main Menu
do_hdf_test:
        call do_hdf                         ;Perform the current
                                            ;Test Case

                                            ;Restore cursor to
                                            ;field that was left
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC>
hdf_test_done:
                                            ;Re-display HDF panel
        DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>
        jmp display_hdf_panel

exit_hdf_pan:

        ret
hdfpan endp

PAGE
;*****************************************************************
;*                                                               *
;*  Procedure Name: DO_HDF                                       *
;*                                                               *
;*  Description  : Performs all the preparation for execution of *
;*                 a call to the Host Data Facility API/CS as well *
;*                 as the call and the necessary housekeeping    *
;*                 following the call to the API/CS.             *
;*                                                               *
;*  Input : Variables from the EZVU II Host Data Facility        *
;*          panel defined in the procedure HDFPAN.               *
;*                                                               *
;*  Output : Return Code, Error Class and Error Type as well     *
;*           as the Offset into the file and the completion byte *
;*           on a status request as well as error messages for   *
;*           invalid input.                                      *
;*                                                               *
;*****************************************************************
do_hdf  proc near
          PUSHREGS                        ;Save all regs
```

```
         mov  ax,delay4                ;Delay requested amount of
         call delay                    ;time

         mov  ax,0ffffh                ;Reset all return codes to ffff

         mov  prime_rc4,ax
         mov  prime_ec4,ax
         mov  prime_et4,ax

         mov     al,reqcode4_asc       ;Convert ascii char input by
         lea     si,hdf_rc_tbl         ;User to binary request code
         call    get_reqcode
         mov     req_code4,ax
;*
;* CONVERT START BYTE
;*
         lea     di,startbyte_asc    ;Convert ascii string input by
         mov     cx,4                ;User to binary start byte

         call    asc2hex

         jcxz    good_startbyte_1    ;Cx = 0 indicates good start byte
         jmp     bad_startbyte

PAGE

good_startbyte_1:
         mov     word ptr startbyte+2,ax

         lea     di,startbyte_asc+4  ;Convert ascii string input by
         mov     cx,4                ;User to binary start byte

         call    asc2hex

         jcxz    good_startbyte_2    ;CX = 0 indicates good START BYTE
         jmp     bad_startbyte

good_startbyte_2:
         mov     word ptr startbyte,ax
;*
;* CALC LENGTH OF PC FILE NAME
;*
         xor  cx,cx
         lea  di,pcfilenm               ;Point DI at the
                                        ;PC file name.

pcfname_loop:
         inc  di                        ;Search for blank to
         inc  cl                        ;determine length of file
         mov  al,[di]                   ;name.
         cmp  al,20h                    ;
         jne  pcfname_loop
         mov  pcflgth,cl                ;Store length in ARB

;*
;* calc length of host file name
;*
         xor  cx,cx                     ;Set CX = 0
         lea  di,hostfilenm             ;Point DI at the
```

```
                                        ;HOST file name.

hostfname_loop:
          inc  di                       ;Search for blank to
          inc  cl                       ;determine length of file
          mov  al,[di]                  ;name.
          cmp  al,20h
          jne  hostfname_loop
          mov  hflgth,cl                 ;Store length in ARB

;*
;* point ax:dx at arb for hdf
;*
          mov     ax,ds                 ;Put segment of arb in ax
          lea     dx,arb_id4            ;Put offset  of arb in dx

PAGE

          mov     ax_reg,ax             ;Save AX and DX for examination
          mov     dx_reg,dx             ;by CHECK_ARB on return
          call    Dcjvd00               ;Call HDF API/CS

          call check_arb                ;Ensure that API/CS found ARB
          mov  arb_found4,al            ;Move result to EZVU display var
          jmp  hdf_done                 ;Process results and exit

bad_startbyte:
          DMPC ISPASM,<LNGTH12PD,PARM12D,EZVU_RC> ;Start byte contains
          jmp  hdf_done                           ;non-hex chars

hdf_done:
;*
;* Convert NEXTBYTE to HEX/ASCII string for display
;*

          mov  ax,word ptr nextbyte      ;Convert first word
          lea  di,nextbyte_asc+4
          call hex2asc

          mov  ax,word ptr nextbyte+2    ;Convert second word
          lea  di,nextbyte_asc
          call hex2asc

;*
;* Convert XFERCOMP to HEX/ASCII string for display
;*

          mov  ah,xfercomp
          lea  di,xfercomp_asc
          call hex2asc

          POPREGS                        ;Restore all regs
          ret

do_hdf  endp

PAGE
```

```
;*****************************************************************
;*                                                               *
;*  Procedure Name: SPCF_MEN_PAN                                 *
;*                                                               *
;*  Description  : Displays a menu of the five different SPCF    *
;*                 functions available through the SPCF API/CS   *
;*                 interface.                                    *
;*                                                               *
;*  Input : The EZVU II variable SPCFOPT from the DCJVCP00 panel. *
;*                                                               *
;*  Output : Loads the panel necessary to execute the selected   *
;*           function.                                           *
;*                                                               *
;*****************************************************************
spcf_men_pan proc near
        PUSHREGS


disp_spcf_menu:
        mov     Active_Keys, PARM19D_KEYS
        call    Set_Active_Keys
        DMPC    ISPASM,<LNGTH19PD,PARM19D,EZVU_RC>  ;Display SPCF menu

        cmp     Zrsp1,F3                       ;Was F3 the exit key?
        jne     not_f3_spmen                   ;No, check next key
        jmp     spcf_men_exit                  ;Yes, return to main menu

not_f3_spmen:
        cmp     Zrsp1,F6                       ;Was F6 the exit key?
        jne     not_f6_spmen                   ;No, check next key
                                               ;Shell out to a
        call    execpgm                        ;secondary command
                                               ;processor.
        dmpc ispasm,<lngth10pd,parm10d,ezvu_rc> ;Reposition cursor upon return
        jmp     disp_spcf_menu                 ;Total redisplay panel

not_f6_spmen:
        cmp     Zrsp2,CR                       ;Was Return the exit key?
        je      choice_run                     ;Yes, process selection
        jmp     unknown_spcf_choice

choice_run:
        mov     reqcode3_asc,'0'               ;Set request code for display
        cmp     spcfopt,1                      ;Was it choice 1?
        jne     choice_gnp                     ;No, check next choice
        call    spcf_run_pan                   ;Yes, run SPCF RUN Panel
        jmp     disp_spcf_menu                 ;Loop back to SPCF menu

choice_gnp:
        cmp     spcfopt,2                      ;Was it choice 2?
        jne     choice_parse                   ;No, check next choice
        call    spcf_gnp_pan                   ;Yes, run SPCF GNP Panel
        jmp     disp_spcf_menu                 ;Loop back to SPCF menu

choice_parse:
        cmp     spcfopt,3                      ;Was it choice 3?
        jne     choice_build                   ;No, check next choice
        call    spcf_parse_pan                 ;Yes, run SPCF PARSE Panel
        jmp     disp_spcf_menu                 ;Loop back to SPCF menu
```

```
choice_build:
        cmp     spcfopt,4                       ;Was it choice 4?
        jne     choice_display                  ;No, check next choice
        call    spcf_build_pan                  ;Yes, run SPCF BUILD Panel
        jmp     disp_spcf_menu                  ;Loop back to SPCF menu

choice_display:
        cmp     spcfopt,5                       ;Was it choice 5?
        jne     choice_puf                      ;No, check next choice
        call    spcf_display_pan                ;Yes, run SPCF DISPLAY Panel
        jmp     disp_spcf_menu                  ;Loop back to SPCF menu

choice_puf:
        cmp     spcfopt,6                       ;Was it choice 6?
        jne     choice_sun                      ;No, check next choice
        call    spcf_puf_pan                    ;Yes, run SPCF PUF panel
        jmp     disp_spcf_menu                  ;Loop back to SPCF menu

choice_sun:
        cmp     spcfopt,7                       ;Was it choice 7?
        jne     choice_ser                      ;No, check next choice
        call    spcf_sun_pan                    ;Yes, run SPCF SUN panel
        jmp     disp_spcf_menu                  ;Loop back to SPCF menu

choice_ser:
        cmp     spcfopt,8                       ;Was it choice 8?
        jne     unknown_spcf_choice             ;No, must be invalid choice
        call    spcf_ser_pan                    ;Yes run SPCF SER panel
        jmp     disp_spcf_menu                  ;Loop back to SPCF menu

unknown_spcf_choice:
        mov  spcfopt,1                          ;Set default choice to 1
        showerr_msg 13                          ;Turn on error msg
                                                ;indicating invalid choice
        jmp  disp_spcf_menu                     ;Loop back to SPCF menu

spcf_men_exit:
              POPREGS
              ret
spcf_men_pan  endp

PAGE

;*****************************************************************
;*                                                               *
;*  Procedure Name: PARSE                                        *
;*                                                               *
;*  Description  : Displays the PARSE panel, DCJVCP01, and calls *
;*                 SPCF PARSE routine.                           *
;*                                                               *
;*  Input : Variables defined for the EZVU II panel.             *
;*                                                               *
;*  Output : Return Code, Error Class and Error Type as well     *
;*           as the Command and Receive Correlator received from *
;*           the host on a receive request as well as error      *
;*           messages for invalid input.                         *
;*                                                               *
```

```
;*****************************************************************
spcf_parse_pan  proc    near
        PUSHREGS
        mov  do_parse_rc,0              ;Init return code for do_parse
        mov  word ptr recid_asc,'  '    ;Clear Parse ID display

                                        ;Clear Parse Corr display
        FILL_CHAR recvcorr_hexasc,'0',asc_corr_length

                                        ;Clear Parse Sense Data display
        FILL_CHAR parse_sense_ascii,'0',parse_sense_ascii_len

tot_disp_parse:
        mov     Active_Keys, PARM25D_KEYS
        call    Set_Active_Keys
        DMPC    ISPASM,<LNGTH25PD,PARM25D,EZVU_RC>  ;Display SPCF PARSE panel

display_parse_panel:
         cmp     Zrspl,F10                      ;Was F10 the exit key?
         jne     not_f10_parse                  ;No, check next key
         jmp     do_parse_test

not_f10_parse:
         cmp     Zrspl,F3                       ;Was F3 the exit key?
         jne     not_f3_parse                   ;No, check next key
         jmp     exit_parse                     ;Yes, Return to SPCF Menu

not_f3_parse:
         cmp     Zrspl,F5                       ;Was F5 the exit key?
         jne     not_f5_parse                   ;No, check next key
         jmp     tot_disp_parse                 ;Yes, Return to SPCF Menu

not_f5_parse:
         cmp     Zrspl,F6                       ;Was F6 the exit key?
         jne     not_f6_parse                   ;No
         jmp     do_2nd_dos_parse               ;Yes

not_f6_parse:
         cmp     Zrspl, F9                      ;Was it display file?
         jne     not_any_parse
         call    spcf_display_pan               ;Yes, display it
         jmp     tot_disp_parse                 ;Then redo panel

not_any_parse:
         jmp     parse_test_done                ;No, redisplay

do_2nd_dos_parse:                               ;Shell out to a
         call    execpgm                        ;secondary command
                                                ;processor.
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC> ;Reposit cursor
         jmp     tot_disp_parse                 ;Total redisplay panel

do_parse_test:
         call do_parse                          ;Perform the current
                                                ;Test Case
         DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC> ;Reposit Cursor

parse_test_done:
         DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>  ;Redisplay panel
         jmp     display_parse_panel

exit_parse:
         POPREGS
         ret
spcf_parse_pan  endp

PAGE

;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_PARSE                                       *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call to the Service Point Command Facility    *
;*                 API/CS as well as the call and the necessary    *
;*                 housekeeping following the call to the API/CS.  *
;*                                                                 *
;*  Input : Variables from the EZVU II SPCF Parse panel.           *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type and Receive   *
;*           Correlator as well as the data parsed from the NMVT   *
;*           The parsed data is placed in an ARB                   *
;*           which is the stored in the specified file.            *
;*           following the file is the Receive Correlator for later *
;*           use.  The parse ARB may be displayed by the Display   *
;*           routine.                                              *
;*                                                                 *
;*                                                                 *
;*******************************************************************
do_parse proc near

          PUSHREGS                        ;Save all regs

          mov  do_parse_rc,0              ;Init return code for do_parse

          mov  ax,0ffffh                  ;Reset all return codes to ffff

          mov  prime_rc6,ax
          mov  prime_ec6,ax
          mov  prime_et6,ax
          mov  prime_rc3,ax
          mov  prime_ec3,ax
          mov  prime_et3,ax

          lea  di,Nmvtfile                ;Point DI at the NMVT file name.
          mov  Filename_Ptr,di            ;Store addr in parm for READ_NMVT
          lea  di,Nmvtbuff                ;Point DI at buffer into which to
                                          ;  read the Alert NMVT.
          mov  Readbuff_Ptr,di            ;Store addr in parm for READ_NMVT
          mov  Readbuff_Size,NMVTBUFF_SIZE    ;Store read buffer size parm
                                              ;for READ_NMVT.

          call Read_Nmvt                ;read in the file
          cmp  Read_Nmvt_Stat,0         ;was read successful?
          je   good_parse_read          ;yes, continue
          jmp  do_parse_bad_read        ;no, exit

good_parse_read:
        mov  ah,byte ptr NMVTbuff       ; Get length of NMVT in ax
        mov  al,byte ptr NMVTbuff+1
        lea  si,NMVTbuff                ; Add length of NMVT to beginning
        add  si,ax                      ; buffer address to get address
                                        ; of correlator read from NMVT file.
                                        ; now ds:[si] points at correlator
                                        ; from NMVT file.

        push ds                         ; Set es = ds as source and dest
        pop  es                         ; are in data segment.

        lea  di,parse_correlator        ; Point es:[di] at target
        mov  cx,10                      ; move 10 bytes
        cld                             ; forward
    rep movsb                           ; move them

        mov  ax,Readbuff_Ptr
        mov  parse_nmvt_offset,ax
        mov  parse_nmvt_segment,ds

        mov     ax,ds                     ;Put segment of arb in ax
        lea     dx,arb_id6                ;Put offset  of arb in dx

        mov     ax_reg,ax                 ;Save AX and DX for examination
        mov     dx_reg,dx                 ;by CHECK_ARB on return

        call    Dcjvb00                   ;CALL THE SPCF API/CS

        call check_arb                    ;Ensure that API/CS found ARB
        mov  arb_found3,al                ;Move result to EZVU display var
        mov  ax, prime_rc6                ;Get return codes into display
        mov  prime_rc3, ax
        mov  ax, prime_ec6                ;Get return codes into display
        mov  prime_ec3, ax
        mov  ax, prime_et6                ;Get return codes into display
        mov  prime_et3, ax

        MOVE_STRING PARSE_CORRELATOR,RECVCORR,10  ; Convert Parse correlator
        call Cnv_Recvcorr                         ; to displayable form.

        mov  di,offset parse_sense_ascii    ; Now do parse sense -
        lea  bx,parse_sense_data            ; requires conversion to ASCII
        mov  cx,parse_sense_data_len

cnv_sense_loop:                             ;Convert the parse sense data
        mov  al, byte ptr [bx]              ;to displayable form.
        call Hexb2asc
        add  di, 2
        inc  bx
        loop cnv_sense_loop

        mov  al,parse_id
        lea  di,recid_asc                 ;Convert Parse ID from ARB to
        call hexb2asc                     ;  ASCII for display

        cmp  prime_rc6,0                  ;Was parse successful ?
        je   check_parse_id               ;yes, continue
```

```
          jmp  do_parse_bad_end           ;no,  exit

check_parse_id:
          cmp  parse_id,061h              ;Is this a RUN command?
          jne  not_run_command            ;No, try others

          xor  cx,cx                      ;Move the command from the API
          mov  cl,parse_command_len       ;buffer to our own buffer

          mov  di,offset parse_data       ;Set es:di to parse_data
          jcxz prs_command_len_zero       ;If command length = 0, jump around

          push ds                         ;Set es = ds
          pop  es

          lds  si,parse_command_ptr       ;Set ds:si to source buffer
          cld
      rep movsb

          push es
          pop  ds                         ;Restore ds from es

prs_command_len_zero:
          mov  ax, PARSE_DATA_OFFSET          ;Reset the pointer for file
          mov  word ptr parse_command_ptr, ax ;Offset is offset to data
          sub  ax, ax                         ;Segment is 0
          mov  word ptr parse_command_ptr+2, ax
          mov  word ptr names_ptr, ax         ;Zero out resource names
          mov  word ptr names_ptr+2, ax       ;Pointer
          jmp do_parse_save                   ;Go save the arb

not_run_command:
          cmp  parse_id,062h              ;Is this a LINK PD command?
          je   link_command               ;Yes, jump around
          cmp  parse_id,063h              ;Is this a LINK DATA command?
          je   link_command               ;Yes, jump around
          cmp  parse_id,064h              ;Is this a LINK TEST command?
          je   link_command               ;Yes, jump around
          jmp  do_parse_bad_end           ;No, must be invalid

link_command:
                                          ;Move names into data area
          xor  cx,cx                      ;Clear count
          mov  bh,no_names                ;How many names

          lea  di,parse_data              ;Target offset

          or   bh,bh                      ;Is number of names = 0 ?
                                          ;  This should never occur, but
                                          ;  is check for just in case.


          jz   no_names_zero              ;Yes, jump around

          push ds                         ;Set target
          pop  es                         ;  segment

          lds  si,names_ptr               ;Source segment and offset ds:si
          cld                             ;Foward
```

```
yet_another_name:
          lodsb                          ;Length of name
          stosb                          ;  into data area
          mov  cl,al                     ;Count of characters
          jcxz zero_len_name             ;If name length = 0, jump around.
                                         ;  This should never occur, but
                                         ;  is check for just in case.
       rep movsb                         ;Move the name.

zero_len_name:
          dec  bh                        ;another name moved
          jnz  yet_another_name

          push es
          pop  ds                        ;restore ds

no_names_zero:
          mov  ax, PARSE_DATA_OFFSET          ;Reset the pointer for file
          mov  word ptr names_ptr, ax         ;Offset is offset to data
          sub  ax, ax                         ;Segment is 0
          mov  word ptr names_ptr+2, ax       ;Zero out command name
          mov  word ptr parse_command_ptr, ax
          mov  word ptr parse_command_ptr+2, ax;Pointer
          jmp do_parse_save                   ;Go save the arb

do_parse_save:
          mov  ax, di                         ;Get present output pointer
          sub  ax, offset arb_id6             ;Calculate length of output

          mov  word ptr Writebuff_Ptr_Tbl,offset arb_id6   ;Set buffer offset
          mov  word ptr Writebuff_Ptr_Tbl+2,ds             ;And buffer segment
          mov  word ptr Writebuff_Ptr_Tbl+4,ax             ;And write size

          mov  word ptr Writebuff_Ptr_Tbl+10,0 ;And mark end of tbl

          mov  Filename_Ptr, offset Arbfile   ;And output name
          call Write_File                     ;And save the output
          cmp  write_file_stat,0              ;Was write successful ?
          jne  do_parse_bad_write             ;no, show error msg
          jmp  do_parse_end                   ;yes exit

do_parse_bad_write:
          SHOWERR_MSG 303        ;Show error msg indicating write failed.
          mov  do_parse_rc,3     ;Set return code to indicate do_parse failed.
          jmp  do_parse_end

do_parse_bad_read:
          SHOWERR_MSG 302        ;Show error msg indicating read failed.
          mov  do_parse_rc,2     ;Set return code to indicate do_parse failed.
          jmp  do_parse_end

do_parse_bad_end:
          SHOWERR_MSG 301        ;Show error msg indicating parse failed.
          mov  do_parse_rc,1     ;Set return code to indicate do_parse failed.
          jmp  do_parse_end

do_parse_end:
```

```
           POPREGS
           ret
do_parse endp

PAGE

;*****************************************************************
;*                                                               *
;*  Procedure Name: spcf_build_pan                               *
;*                                                               *
;*  Description  : Displays the BUILD panel, DCJVCP02, and calls *
;*                  SPCF BUILD routine.                          *
;*                                                               *
;*  Input : Variables defined for the EZVU II panel.             *
;*                                                               *
;*  Output : Return Code, Error Class and Error Type as well     *
;*           as the Command and Receive Correlator received from *
;*           the host on a receive request as well as error      *
;*           messages for invalid input.                         *
;*                                                               *
;*****************************************************************
spcf_build_pan  proc    near
        PUSHREGS
        mov  ax,0ffffh                  ;RESET ALL RETURN CODES TO FFFF
        mov  prime_rc3,ax
        mov  prime_ec3,ax
        mov  prime_et3,ax

tot_disp_build:
        mov     Active_Keys, PARM26D_KEYS
        call    Set_Active_Keys
        DMPC    ISPASM,<LNGTH26PD,PARM26D,EZVU_RC>  ;Display SPCF BUILD panel

display_build_panel:
         cmp     Zrspl,F10                   ;Was F10 the exit key?
         jne     not_f10_build               ;No, check next key
         jmp     do_build_test

not_f10_build:
         cmp     Zrspl,F3                    ;Was F3 the exit key?
         jne     not_f3_build                ;No, check next key
         jmp     exit_build                  ;Yes, Return to SPCF Menu

not_f3_build:
         cmp     Zrspl,F5                    ;Was F5 the exit key?
         jne     not_f5_build                ;No, check next key
         jmp     tot_disp_build              ;Yes, Return to SPCF Menu

not_f5_build:
         cmp     Zrspl,F6                    ;Was F6 the exit key?
         jne     not_f6_build                ;No, check next
         jmp     do_2nd_dos_build            ;Yes
                                             ;the panel.
not_f6_build:
         cmp     Zrspl, F9                   ;Was it display file?
         jne     not_any_build
         call    spcf_display_pan            ;Yes, display it
         jmp     tot_disp_build              ;Then Return to SPCF Menu
```

```
not_any_build:
        jmp     build_test_done                 ;No, redisplay


do_2nd_dos_build:                               ;Shell out to a
        call    execpgm                         ;secondary command
                                                ;processor.
        jmp     tot_disp_build                  ;Total redisplay panel

do_build_test:
        call do_build                           ;Perform the current
                                                ;Test Case
        DMPC ISPASM,<LNGTH10PD,PARM10D,EZVU_RC> ;Reposit Cursor

build_test_done:
        DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>  ;Redisplay panel
        jmp     display_build_panel

exit_build:

        POPREGS
        ret
spcf_build_pan  endp


PAGE


;*******************************************************************
;*                                                                 *
;*  Procedure Name: DO_BUILD                                       *
;*                                                                 *
;*  Description  : Performs all the preparation for execution of   *
;*                 a call to the Service Point Command Facility    *
;*                 API/CS as well as the call and the necessary    *
;*                 housekeeping following the call to the API/CS.  *
;*                                                                 *
;*  Input : Variables from the EZVU II SPCF Parse panel.           *
;*                                                                 *
;*  Output : Return Code, Error Class and Error Type and Receive   *
;*           Correlator as well as the data parsed from the NMVT   *
;*           The parsed data is placed in an ARB                   *
;*           which is the stored in the specified file.            *
;*           following the file is the Receive Correlator for later *
;*           use.  The parse ARB may be displayed by the Display   *
;*           routine.                                              *
;*                                                                 *
;*                                                                 *
;*******************************************************************

;***********************************************************************
; fix_seg
;
; Input:
;      segptr  - An Intel DD type
;      buffer  - Buffer where the data is (offset added to offset in
;                   buffer)
;      badexit - Place to jump to if address out of range of buffer
;
```

```
; Output:
;       segptr  - segment portion fixed up
;
; Also uses bd_datsize in checking things
;*************************************************************************
FIX_SEG         macro  segptr,  buffer, badexit
        local   check_segment                   ;Bad exit
        local   all_ok                          ;Good exit

        mov     ax, word ptr segptr             ;Get offset
        cmp     ax, bd_datsize                  ;Make sure it is in range
        jle     check_segment                   ;If not, exit
        jmp     badexit                         ;Else go to bad exit
check_segment:
        add     ax, offset buffer               ;Add buffer offset
        mov     word ptr segptr, ax             ;And store it back
        cmp     word ptr segptr+2, 0            ;Make sure segment was 0
        je      all_ok
        jmp     badexit
all_ok:
        push    ds
        pop     word ptr segptr+2
        endm

PAGE


;*************************************************************************
; do_build
;
; Reads a build ARB from a file and converts it into an NMVT
;
; Input:
;
;       Arbfile         - Name of file to be read in
;       Nmvtfile        - Name of file to write NMVT to
;
; Output:
;
;       Arbbuff         - Contains ARB read in
;       bd_rx           - Set to 0 if no error, nonzero otherwise
;                               (set when errors in ARB in file found)
;
;       The following are set to the build API return codes, if the ARB
;               was not found invalid prior to the call to the API
;       PRIME_RC3           - Set to 0 if no error, nonzero otherwise
;       PRIME_EC3           - Set to 0 if no error, nonzero otherwise
;       PRIME_EC3           - Set to 0 if no error, nonzero otherwise
;
;*************************************************************************
do_build        proc near
        PUSHREGS                        ;save registers
        nop
        nop
        mov     arb_found3,' '          ;Clear ARB found variable
        mov     ax, BD_NOERR
        mov     bd_rx, ax               ;Clear internal return code
        mov     ax, 0ffffh              ;Initialize external return codes
        mov     Arbbuff.bd_retcode,ax
        mov     Arbbuff.bd_errclass,ax
        mov     Arbbuff.bd_errtype,ax

;*********************************************************************
; Read the file in, checking for errors
;*********************************************************************
        mov     filename_ptr, offset arbfile     ;Set up file name
        mov     readbuff_ptr, offset Arbbuff     ;Set up buffer address
        mov     readbuff_size, bd_bufsize        ;And the buffer size
        call    read_nmvt                        ;Read stuff in
        cmp     read_nmvt_stat, 0                ;Check for success
        je      bd_read_ok                       ;If ok, continue
        mov     ax,read_nmvt_stat                ;Get error code
        mov     bd_rx, ax                        ;And use that as our rc
        jmp     exit_do_build                    ;and leave

;*********************************************************************
; File read in OK, check to make sure we recognize it as an ARB
;*********************************************************************
bd_read_ok:
        mov     ax, 0ffffh              ;Initialize external return codes
        mov     Arbbuff.bd_retcode,ax
        mov     Arbbuff.bd_errclass,ax
        mov     Arbbuff.bd_errtype,ax
        mov     ax, filesize                    ;Get the file size
        mov     bd_datsize, ax                  ;And store it

;Now, check the ARB ID

        COMPARE_STRINGS Arbbuff.bd_arbid,bd_refarb.bd_arbid, BD_ARBIDLEN
        je      bdcheck_reqcode         ;If ok, check request code
        mov     bd_rx, BD_ARBIDERR      ;Else set error code
        jmp     build_error_exit

;Check request code
bdcheck_reqcode:
        mov     ax, bd_refarb.bd_reqcode;Get reference request code
        cmp     Arbbuff.bd_reqcode, ax  ;Compare it to one read in
        je      bdcheck_arblen          ;If ok, check length
        mov     bd_rx, BD_REQCDERR      ;Else set up error code
        jmp     build_error_exit

;Check the ARB length
bdcheck_arblen:
        mov     al, Arbbuff.bd_arblen   ;Get arb length
        cmp     al, bd_refarb.bd_arblen ;And check it
        je      bdcheck_buildid         ;OK, check the build ID
        mov     bd_rx, BD_ARBLENERR     ;No good, set up error code
        jmp     build_error_exit

;Check to make the the build ID is 62 through 64
bdcheck_buildid:
        mov     al, Arbbuff.bd_buildid  ;Get build id
        cmp     al, BD_ID_LPD           ;Was it LINK PD?
        jne     bdcheck_linkdata        ;No, check for link data
        mov     bd_type, BD_TYPE0       ;Set type to LINK PD?
        mov     bd_lccdclen, BD_LCCDCLEN0
        jmp     bd_start                ;Start real work
```

```
;Was not LINKPD - make sure it is LINKDATA or LINKTEST
bdcheck_linkdata:
        cmp     ax, BD_ID_LD            ;Was it LINKDATA?
        je      bdcheck_linkok          ;If so, is ok
        cmp     ax, BD_ID_LT            ;Else check for LINKTEST
        je      bdcheck_linkok          ;If so, is ok
        jmp     passthrough             ;Else just pass the ARB through
bdcheck_linkok:
        mov     bd_type, BD_TYPE1       ;Was LINKDATA or LINKTEST
        mov     bd_lccdclen, BD_LCCDCLEN1

;*********************************************************************
; Now we are reasonable sure the stuff we read in was a build ARB,
;       so we can start doing the fixup
;*********************************************************************
bd_start:
;First, fix up the segment of the probable cause pointer
        mov     bd_rx,BD_PCAUSERR       ;Set up error condition
        FIX_SEG Arbbuff.bd_probcause, Arbbuff, address_error

;Now, fix put the path list info pointer
        mov     bd_rx,BD_PLISTERR       ;Set up error condition
        FIX_SEG Arbbuff.bd_pathlist, Arbbuff, address_error

;Make bx point to path information list control block
        mov     bx, word ptr Arbbuff.bd_pathlist
        mov     bd_rx,BD_PLCBERR        ;Set up error condition
        FIX_SEG bdlcc_ptr[bx], Arbbuff, address_error     ;Fix up segment

        mov     bd_rx, BD_NOERR         ;Reset error code
        cmp     bd_type, BD_TYPE0       ;See if it is LINKPD
        jne     checkcount              ;If not, do some more
        jmp     passthrough             ;********* JHC

checkcount:
        mov     cx, bdlcc_num[bx]       ;Set cx to number of LCC things
        cmp     cx, 0                   ;Was it zero?
        jne     lccdcstart              ;If not, do processing
        jmp     passthrough             ;******** JHC

;*********************************************************************
; Now go through the lcc description control blocks
;*********************************************************************
lccdcstart:
        mov     bx, word ptr bdlcc_ptr[bx]      ;Set bx to start of array

lccdcbloop:
        push    cx                      ;Save our loop counter
        push    bx                      ;And our pointer
        mov     bd_rx, BD_LCCDCERR      ;Set return code
        FIX_SEG bdlcc_dataptr[bx], Arbbuff, baddc ;Fix up segment

        mov     cx,bdlcc_number[bx]     ;Get number of data elements
        cmp     cx,0                    ;See if its more than one
        jne     lccdbstart              ;If so, do processing
        jmp     gooddc                  ;Else continue with dc blocks

;*********************************************************************
; Loop for data elements in a single control block
```

```
;*************************************************************************
lccdbstart:
        mov     bx, word ptr bdlcc_dataptr[bx]  ;Bx now points to data array
lccdbloop:
        mov     bd_rx, BD_LCCDBERR      ;Set return code
        FIX_SEG bdlcc_dvptr[bx], Arbbuff, baddc   ;Fix up segment
        mov     al, bdlcc_dnlen[bx]     ;Get name length
        sub     ah, ah                  ;Zero out high byte
        add     ax, BD_LCCDBLEN         ;And overhead length to get to next
        add     bx, ax                  ;Then add it all to offset
        loop    lccdbloop               ;And loop until done
        jmp     gooddc                  ;Don't want to think we broke

;*************************************************************************
; End of loop for array of control blocks
;*************************************************************************
baddc:  pop     bx
        pop     cx
        jmp     address_error
gooddc: pop     bx
        pop     cx
        add     bx, bd_lccdclen         ;Go to next array element
        loop    lccdcbloop

;*************************************************************************
; Call the build API
;*************************************************************************
passthrough:
        push    ds                      ;Set AX:DX to point to ARB
        pop     ax
        mov     dx, offset Arbbuff
        PUSHREGS
        mov     ax_reg,ax               ;Save AX and DX for examination
        mov     dx_reg,dx               ;by CHECK_ARB on return

        call    Dcjvb00                 ;Call the build procedure
        call    check_arb               ;Ensure that API/CS found ARB
        mov     arb_found3,al           ;Move result to EZVU display var
        POPREGS
        cmp     Arbbuff.bd_retcode, 0    ;Check the return code
        jne     bd_builderr
        mov     ax, Arbbuff.bd_builtnmvtlen
        cmp     ax,0                            ;Was created NMVT length 0?
        jne     do_write_nmvt                   ;If not, write it out
        jmp     bd_builderr                     ;Else exit
;*************************************************************************
; Write the NMVT out to the file
;*************************************************************************
do_write_nmvt:
        mov     filename_ptr, offset Nmvtfile

        les     bx, Arbbuff.bd_builtnmvt        ;Get NMVT address in ES:BX
        mov     word ptr Writebuff_Ptr_Tbl,bx   ;Save the offset
        mov     word ptr Writebuff_Ptr_Tbl+2,es ;And the segment
        mov     word ptr Writebuff_Ptr_Tbl+4,ax ;And the NMVT length

        lea     bx,Arbbuff.bd_correlator           ;
        mov     word ptr Writebuff_Ptr_Tbl+6,bx    ; Save the offset
        mov     word ptr Writebuff_Ptr_Tbl+8,ds    ; And the segment
        mov     word ptr Writebuff_Ptr_Tbl+10,10   ; And the correlator length

        mov     word ptr Writebuff_Ptr_Tbl+16,0    ; Mark end of table

        call    Write_File             ;All OK, so save NMVT
        jmp     exit_do_build

bd_builderr:
        mov     bd_rx, BD_BLDERR       ;Set up our return code
        jmp     build_error_exit

build_error_exit:
address_error:
        SHOWERR_MSG bd_rx

exit_do_build:
        push    Arbbuff.bd_retcode     ;Set up standard return codes
        pop     PRIME_RC3
        push    Arbbuff.bd_errclass
        pop     PRIME_EC3
        push    Arbbuff.bd_errtype
        pop     PRIME_ET3
        POPREGS
        ret
do_build        endp

PAGE

        INCLUDE  APIMAIN.UTL               ;Utility subroutines

;* Make all utility routines public
;*

PUBLIC CNV_RECVCORR
PUBLIC CNV_SENDCORR
PUBLIC LOAD_SENDCORR
PUBLIC SAVE_RECVCORR
PUBLIC DEL_SENDCORR
PUBLIC CLEAR_CORR_TBL
PUBLIC GET_REQCODE
PUBLIC CHECK_ARB

PAGE

PUBLIC CHOICE              ;State variable for SELMENU
PUBLIC Zrsp1               ;Scan  code of key that caused Panel Exit
PUBLIC Zrsp2               ;ASCII code of key that caused Panel Exit
PUBLIC Zent1               ;Scan code of key to be used as Enter key
PUBLIC Zent2               ;ASCII code of key to be used as Enter key
PUBLIC Zent1a              ;Scan code of key to be used as Enter key
PUBLIC Zent2a              ;Scan code of key to be used as Enter key
PUBLIC Zent1b              ;Scan code of key to be used as Enter key
PUBLIC Zent2b              ;Scan code of key to be used as Enter key
PUBLIC Zent1c              ;Scan code of key to be used as Enter key
PUBLIC Zent2c              ;Scan code of key to be used as Enter key
PUBLIC Zent1e              ;Scan code of ESC key
PUBLIC Zent2e              ;ASCII code of ESC key
PUBLIC Zent1f              ;F4 key - scan  code
PUBLIC Zent2f              ;F4 key - ASCII code
PUBLIC Zent1n              ;Return key - scan  code
PUBLIC Zent2n              ;Return key - ASCII code
PUBLIC Zatr                ;Color used when input field is highlighted
                           ;Ebony foreground, white background

PUBLIC Zent1PUP            ;Scan code of PgUp key
PUBLIC Zent2PUP            ;ASCII code of PgUp key

PUBLIC Zent1PDN            ;Scan code of PgUp key
PUBLIC Zent2PDN            ;ASCII code of PgUp key

PUBLIC LNGTH9V             ;Number of places for extra enter keys
PUBLIC ioretcod            ;File I/O return code for error messages.


CSEG    ENDS

        END     START
```

APIMAIN.UTL

```
PAGE
;        API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;        SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.

;*********************************************************************
;*                                                                   *
;*  Procedure Name: CNV_RECVCORR                                     *
;*                                                                   *
;*  Description  : Converts the 10 byte binary receive correlator   *
;*                 sent by the SPCF API/CS in response to a receive *
;*                 request to a 20 char Hex/ASCII string so that it *
;*                 may be displayed in a human readable form.       *
;*                                                                   *
;*  Input : The 10 byte binary receive correlator - RECVCORR        *
;*                                                                   *
;*  Output : The 20 char Hex/ASCII string to be displayed on        *
;*           the SPCF panel - RECVCORR_HEXASC                        *
;*                                                                   *
;*********************************************************************
PUBLIC Cnv_Recvcorr
Cnv_Recvcorr proc near
             pushregs                    ;Save all regs
;*
;* Receive correlator is 10 bytes long and since HEX2ASC can convert
;* two bytes at a time init loop count to 5
;*
             mov  cx,asc_corr_length/4    ;ASC_CORR_LENGTH = 20
             lea  di,recvcorr_hexasc      ;Point DI at Hex/ASCII string buffer
```

```
            lea  si,recvcorr            ;Point SI at binary Receive correlator

cnv_recv_loop:
;*
;* Must load bytes one at a time to avoid the byte swapping
;* that would be caused by loading them as a word.
;*
            mov  ah, byte ptr [si]      ;Load left most byte
            mov  al, byte ptr [si+1]    ;Load right byte
            call hex2asc                ;Convert 2 bytes
            add  si,2                   ;Bump string pointer
            add  di,4                   ;Bump binary pointer
            dec  cx                     ;Decrement loop count
            jcxz cnv_recv_done          ;Is conversion complete?
            jmp  cnv_recv_loop          ;No, convert next 2 bytes.

cnv_recv_done:
            popregs                     ;Restore all regs
            ret

Cnv_Recvcorr endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: Cnv_Sendcorr                                   *
;*                                                                 *
;*  Description  : Converts the 20 char Hex/ASCII string inputted  *
;*                 by the user to a 10 binary byte string in the   *
;*                 form expected by the SPCF API/CS.               *
;*                                                                 *
;*  Input : The 20 char Hex/ASCII string inputted by the user -    *
;*          SENDCORR_ASC                                           *
;*                                                                 *
;*  Output : The 10 binary byte string to be passed to the SPCF    *
;*           API/CS for a Send request.                            *
;*           If the string inputted by the user contained any chars *
;*           other than '0-9' or 'A-F' the conversion will fail.   *
;*           On return from this routine the variable SENDCORR_STAT *
;*           will be set to zero if the conversion was successful  *
;*           and set to hex FF if the conversion failed.           *
;*                                                                 *
;*******************************************************************
PUBLIC Cnv_Sendcorr
Cnv_Sendcorr proc near
            pushregs                    ;Save all regs

            mov  sendcorr_stat,0        ;Init conversion status to good

;*
;* Send correlator in Hex/ASCII format is 20 bytes long and since
;* ASC2HEX can convert up to four bytes at a time init loop count to 5
;*
            mov  sendcorr_cnt,asc_corr_length/4
            lea  di,sendcorr_hexasc     ;Point DI at Hex/ASCII input string
            lea  si,sendcorr            ;Point SI at Binary output buffer

cnv_send_loop:
            mov  cx,4                   ;Length of string to be converted = 4
            call asc2hex                ;Do the conversion
            cmp  cx,-1                  ;Did the conversion fail?
            jne  good_sendcorr_cnv      ;No, converted good
            jmp  bad_sendcorr_cnv       ;Yes, conversion failed, exit

good_sendcorr_cnv:
            mov  byte ptr [si]  ,ah     ;Put converted bytes in
            mov  byte ptr [si+1],al     ;binary buffer
            add  si,2                   ;Bump binary pointer
            add  di,4                   ;Bump string pointer
            dec  sendcorr_cnt           ;Decrement loop counter
            cmp  sendcorr_cnt,0         ;Conversion complete, Y/N?
            je   cnv_send_done          ;Yes
            jmp  cnv_send_loop          ;No, convert next 4 chars

PAGE

bad_sendcorr_cnv:
            mov  sendcorr_stat,0ffh     ;Indicate bad conversion
            DMPC ISPASM,<LNGTH17PD,PARM17D,EZVU_RC>   ;Display error msg
                                                      ;indicating non-hex
                                                      ;chars in input fld
            jmp  cnv_send_done

cnv_send_done:
            popregs                     ;Restore all regs
            ret

Cnv_Sendcorr endp


;*****************************************************************
;*                                                               *
;*  Procedure Name: load_sendcorr                                *
;*                                                               *
;*  Description   : Displays a panel showing a list of all       *
;*                  of the correlators to which no response has  *
;*                  been sent. The user may select any correlator *
;*                  or press Esc to return without making a      *
;*                  selection.                                   *
;*                                                               *
;*  Table structure: The correlator table actually consists of   *
;*                   two related tables. The first is a table of *
;*                   MAX_CORR_CNT one byte entries that indicate *
;*                   whether the corresponding record in the ASCII *
;*                   table is in use. FF hex means the corresponding*
;*                   record in the ASCII table is empty. Any other *
;*                   value indicates the corresponding correlator's *
;*                   place in the list with a number 1 -         *
;*                   MAX_CORR_CNT,where the largest number is the *
;*                   most recently received correlator. The ASCII *
;*                   table is a series of MAX_CORR_CNT ASCII     *
;*                   records each ASC_CORR_LENGTH long, each of  *
;*                   which is a correlator.                      *
;*                                                               *
;*****************************************************************
load_sendcorr proc near
        pushregs
        cmp     unresponded_cnt,0               ; Are there any outstanding
                                                ; correlators ?
        jne     outstanding_exist               ; yes, continue
        mov     ax,211                          ; no, show informational msg
        call    show_errmsg                     ; and
        jmp     load_sendcorr_exit              ; exit subrout

outstanding_exist:

;*
;*  The following High level code describes the sorting algorithm used to
;*  sort the correlator table in the order received.
;*
;*  SORTFLAG = 1
;*  DO WHILE SORTFLAG = 1
;*     SORTFLAG = 0
;*     DO I = 1 TO MAX_CORR_CNT-1
;*        IF CORR_RANK_TBL(I) > CORR_RANK_TBL(I+1) THEN
;*           DO;
;*             SWAP CORR_RANK_TBL(I) WITH CORR_RANK_TBL(I+1)
;*             SWAP CORR_ASC_TBL (I) WITH CORR_ASC_TBL (I+1)
;*             SORTFLAG = 1
;*           END;
;*     END;
;*  END;
;*
        mov sortflag,1                          ;sortflag = 1

while_sortflag_1:
        cmp sortflag,1                          ;do while sortflag = 1
        je  continue_sort
        jmp sort_done

continue_sort:
          mov sortflag,0                        ;SORTFLAG = 0
          mov cx,max_corr_cnt-1
          xor di,di                             ;Zero index into CORR_RANK_TBL
          xor bx,bx                             ;Zero index into CORR_ASC_TBL

for_i_1_to_max_corr_cnt_1s1:
            mov ax, word ptr corr_rank_tbl[di]  ;(I)th   Entry in AL
                                                ;(I+1)th Entry in AH

            cmp al,ah                           ;Are Entries in order ?
            ja  swap_entries                    ;No, swap them
            jmp dont_swap                       ;Yes, bump to next entries

swap_entries:

;*
;* Swap the (I)th and (I+1)th entries in the Rank table
;*
              mov  dh,al
              mov  dl,ah
              mov  word ptr corr_rank_tbl[di],dx
```

```
;*
;* Swap the (I)th and (I+1)th entries in the ASCII table
;*
  MOVE_STRING CORR_ASC_TBL[BX]TEMPCORR_HEXASC,ASC_CORR_LENGTH
  MOVE_STRING CORR_ASC_TBL[BX+ASC_CORR_LENGTH],CORR_ASC_TBL[BX],ASC_CORR_LENGTH
  MOVE_STRING TEMPCORR_HEXASC,CORR_ASC_TBL[BX+ASC_CORR_LENGTH],ASC_CORR_LENGTH

;*
;* Indicate a swap occured to force another
;* pass through the WHILE_SORTFLAG_1 Loop.
;*
            mov  sortflag,1

dont_swap:
          inc  di                       ;Point at next entry in CORR_RANK_TBL
          add  bx,asc_corr_length       ;Point at next entry in CORR_ASC_TBL

        loop for_i_1_to_max_corr_cnt_ls1  ;End of FOR I = 1 to MAX_CORR_CNT -1

      jmp while_sortflag_1              ;End of WHILE SORTFLAG = 1

sort_done:
      mov     Active_Keys, PARM24D_KEYS
      call    Set_Active_Keys               ;Set allowed keys
      DMPC    ISPASM,<LNGTH24PD,PARM24D,EZVU_RC>  ;Display CORR menu

check_corr_option:
      cmp     zrspl,f3                      ;Was F3 the exit key?
      jne     not_f3_key                    ;No check the next key
      jmp     load_sendcorr_exit            ;Yes, exit subrout

not_f3_key:
      cmp     zrsp2,esc                     ;Was ESC the exit key?
      jne     not_esc_corrmen               ;No, check next key
      jmp     load_sendcorr_exit            ;Yes, exit subrout

not_esc_corrmen:
      cmp     zrsp2,cr                      ;Was Return the exit key?
      je      corr_selected                 ;Yes, process selection
      jmp     unknown_corr_choice

corr_selected:                              ; JOF 7-24-87
      cmp     corropt,0                     ;Is correlator selected 0 ?
      je      invalid_corropt               ;Yes, show error & redisplay

not_0_corropt:
      mov     al,unresponded_cnt
      cmp     corropt,al              ;Is correlator selected valid ?
      ja      invalid_corropt         ;No, show error & redisplay
      jmp     valid_corropt           ;Yes

invalid_corropt:
      showerr_msg 24                  ;Invalid entry error msg
      jmp     unknown_corr_choice

unknown_corr_choice:
      DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>  ;Display CORR menu again
      jmp     check_corr_option
```

```
valid_corropt:
        xor ah,ah                 ;Calculate displacement into CORR_ASC_TBL
        mov al,corropt            ;of selected correlator.
        dec ax                    ;Disp = (Selected Num - 1) * ASC_CORR_LENGTH
        mul by_asc_corr_length
        mov bx,ax                 ;Put displacement in BX

; Move selected correlator into Send correlator buffer
        MOVE_STRING CORR_ASC_TBL[BX],SENDCORR_HEXASC,ASC_CORR_LENGTH

        jmp load_sendcorr_exit

load_sendcorr_exit:
              mov     ax,word ptr zentlf          ;Restore F4 as enter key
              mov     word ptr zentl,ax
              popregs                             ;Restore regs
              ret
load_sendcorr endp

PAGE

PUBLIC save_recvcorr
save_recvcorr proc near
;*******************************************************************
;*                                                                 *
;*  Procedure Name: save_recvcorr                                  *
;*                                                                 *
;*  Description  : Searches the correlator table for an            *
;*                 empty record amd stores the current number of   *
;*                 receive correlators to which no reply has been  *
;*                 sent in CORR_RANK_TBL and the ASCII             *
;*                 representation of the receive correlator being  *
;*                 saved in the corresponding entry of CORR_ASC_TBL.*
;*                                                                 *
;*  Input: RECVCORR_HEXASC - ASCII representation of Receive       *
;*                           correlator received from host.        *
;*                                                                 *
;*  Output: - Current Receive correlator along with its sequence   *
;*            number is stored in CORR_TBL.                        *
;*        UNRESPONDED_CNT - Number of receive correlators to       *
;*                          which no response has been sent        *
;*                          is incremented.                        *
;*                                                                 *
;*  Table structure: The correlator table actually consists of     *
;*                   two related tables. The first is a table of   *
;*                   MAX_CORR_CNT one byte entries that indicate   *
;*                   whether the corresponding record in the ASCII *
;*                   table is in use. FF hex means the corresponding*
;*                   record in the ASCII table is empty. Any other *
;*                   value indicates the corresponding correlator's *
;*                   place in the list with a number 1 -           *
;*                   MAX_CORR_CNT,where the largest number is the  *
;*                   most recently received correlator. The ASCII  *
;*                   table is a series of MAX_CORR_CNT ASCII       *
;*                   records each ASC_CORR_LENGTH long, each of    *
;*                   which is a correlator.                        *
;*                                                                 *
;*******************************************************************
              pushregs
```

```
            mov cx,max_corr_cnt           ;Set loop count to max
                                          ;number of records.

            xor bx,bx                     ;Set displacement into
                                          ;ASCII table to first record.

            xor di,di                     ;Set displacement into
                                          ;rank table to first record.

srch4_slot:
            cmp  corr_rank_tbl[di],0ffh   ;Is current record empty ?
            je   slot_found               ;Yes

            inc  di                          ;No, Bump DI to next rank record
            add  bx,asc_corr_length          ;    Point BX at next ASCII rec
            loop srch4_slot                  ;Any more records ?

            showerr_msg 22                   ;Error msg - table full
            jmp save_recvcorr_exit           ;exit subrout

slot_found:
            inc unresponded_cnt           ;Increment count of number of
                                          ;receive correlators to which
                                          ;no response has been sent.
;*
;* Put number of receive correlators to which no response has been sent
;* in the first byte of the record. This is also this correlator's
;* sequential rank in the table.
;*
            mov  al,unresponded_cnt
            mov  corr_rank_tbl[di],al
;*
;* Put ASCII version of correlator from host in record.
;*
            MOVE_STRING RECVCORR_HEXASC,CORR_ASC_TBL[BX],ASC_CORR_LENGTH

save_recvcorr_exit:
            popregs
            ret
save_recvcorr endp

PAGE

PUBLIC del_sendcorr
del_sendcorr proc near
;*******************************************************************
;*                                                                 *
;*  Procedure Name: del_sendcorr                                   *
;*                                                                 *
;*  Description  : Searches the correlator table for the           *
;*                 current send correlator and deletes it from     *
;*                 CORR_RANK_TBL and CORR_ASC_TBL.                 *
;*                                                                 *
;*  Input: SENDCORR_HEXASC - ASCII representation of Receive       *
;*                           correlator to be deleted from the     *
;*                                                                 *
;*  Output: - Current Send correlator along with its sequence      *
;*           number is deleted from CORR_ASC_TBL and            *
;*           CORR_RANK_TBL.                                     *
;*                                                              *
;*           UNRESPONDED_CNT - Number of receive correlators to *
;*                             which no response has been sent  *
;*                             is decremented.                  *
;*                                                              *
;*  Table structure: The correlator table actually consists of  *
;*                  two related tables. The first is a table of *
;*                  MAX_CORR_CNT one byte entries that indicate *
;*                  whether the corresponding record in the ASCII *
;*                  table is in use. FF hex means the corresponding*
;*                  record in the ASCII table is empty. Any other *
;*                  value indicates the corresponding correlator's *
;*                  place in the list with a number 1 -         *
;*                  MAX_CORR_CNT,where the largest number is the *
;*                  most recently received correlator. The ASCII *
;*                  table is a series of MAX_CORR_CNT ASCII      *
;*                  records each ASC_CORR_LENGTH long, each of   *
;*                  which is a correlator.                      *
;*                                                              *
;***************************************************************
             pushregs
             mov inact_corr,0

             cmp prime_rc3,0
             je  yes_del_corr

             cmp prime_rc3,8
             je  check_ec3
             jmp no_dont_del_corr
check_ec3:
             cmp prime_ec3,23
             je  check_et3
             jmp no_dont_del_corr
check_et3:
             cmp prime_et3,65
             jne no_dont_del_corr
             mov inact_corr,1
             jmp yes_del_corr

no_dont_del_corr:
             jmp del_sendcorr_exit

yes_del_corr:
             mov cx,max_corr_cnt          ;Set loop count to max
                                          ;number of records.

             xor bx,bx                    ;Set displacement into
                                          ;ASCII table to first record.

             xor di,di                    ;Set displacement into
                                          ;rank table to first record.

srch4_corr_match:
             cmp  corr_rank_tbl[di],0ffh  ;Is current record empty ?
             jne  compare_corrs           ;No
             jmp  empty_corr              ;Yes
;*
```

```
;* No, compare SENDCORR_HEXASC to current table entry.
;*

compare_corrs:
              COMPARE_STRINGS SENDCORR_HEXASC,CORR_ASC_TBL[BX],ASC_CORR_LENGTH
              je   corrs_match
              jmp  corrs_dont_match

corrs_match:
              dec unresponded_cnt           ;Decrement count of number of
                                            ;receive correlators to which
                                            ;no response has been sent.


;*
;* Mark current rank record empty.
;*
              mov  al,corr_rank_tbl[di]            ;Save rank of deleted corr
              mov  corr_deleted,al
              mov  corr_rank_tbl[di],0ffh          ;Mark deleted correlators
                                                   ;rank entry as empty.
;*
;* Blank out current ASCII record.
;*
              FILL_CHAR CORR_ASC_TBL[BX],' ',ASC_CORR_LENGTH

              cmp inact_corr,0
              je  no_inact_msg
              showerr_msg 33

no_inact_msg:
              mov cx,max_corr_cnt           ;Set loop count to max
                                            ;number of records.
              xor di,di                     ;Set displacement into
                                            ;rank table to first record.
rank_adjust_loop:
              cmp  corr_rank_tbl[di],0ffh   ;Is current record empty ?
              je   no_adjust_rank_entry     ;Yes
              mov  al,corr_deleted          ;No
              cmp  corr_rank_tbl[di],al     ;Is current record rank > deleted
                                            ;records rank ?
              jb   no_adjust_rank_entry
              dec  corr_rank_tbl[di]

no_adjust_rank_entry:
              inc  di
              loop rank_adjust_loop

              jmp del_sendcorr_exit            ;Exit subrout

corrs_dont_match:
empty_corr:
              inc  di                       ;Bump DI to next rank record
              add  bx,asc_corr_length       ;Point BX at next ASCII rec
              dec  cx                      ;
              jcxz corr_not_found           ;Any more records ?

              jmp  srch4_corr_match         ;Yes, check next record.
```

```
corr_not_found:
              showerr_msg 25                  ;Error msg - Match not found
              jmp del_sendcorr_exit           ;exit subrout

del_sendcorr_exit:
              popregs
              ret
del_sendcorr endp

;*
;* Description : After a successful close of SPCF this subrout is called
;*               to clear the correlator table.
;*
;* Input : None
;*
;* Output : Clears CORR_RANK_TBL and CORR_ASC_TBL and zeroes UNRESPONDED_CNT
;*

PUBLIC clear_corr_tbl
clear_corr_tbl proc near
;*
;* Set all rank entries to empty value FFH
;*
               FILL_CHAR CORR_RANK_TBL,0FFH,MAX_CORR_CNT
;*
;* Blank out all ASCII correlator entries.
;*
               FILL_CHAR CORR_ASC_TBL,' ',CORR_ASC_TBL_LENGTH
;*
;* Good close clears all outstanding correlators, so zero count.
;*
               mov unresponded_cnt,0
               ret
clear_corr_tbl endp


;*******************************************************************
;*                                                                 *
;*  Procedure Name: GET_REQCODE                                    *
;*                                                                 *
;*  Description  : Converts the single character Request Codes     *
;*                 input by the user to the hexadecimal code       *
;*                 that must be placed in the ARB to perform the   *
;*                 selected request.                               *
;*                                                                 *
;*  Input : Register AL contains the single char inputted by the   *
;*          user.                                                  *
;*          Register SI points at the table to be used to perform  *
;*          the conversion.                                        *
;*                                                                 *
;*  Output : The AX register is used to return the Hex word that   *
;*           is the request code. If the char passed to this       *
;*           routine in AL is not found in the conversion table    *
;*           AX will be set to zero.                               *
;*                                                                 *
;*  Table structure: The table used to perform consists of a series *
;*                   of records of length 3 terminated by a record  *
;*                  whose first byte is '*'. The first byte of     *
;*                  each record is a possible ASCII value enter    *
;*                  by the user on an EZ-VU II panel. The next two *
;*                  bytes contain the one word hex value that      *
;*                  corresponds to that ASCII value.               *
;*                                                                 *
;*******************************************************************
get_reqcode proc near
reqcode_loop:
            mov  ah,byte ptr [si]      ;Put ASCII value from table in AH
            cmp  ah,al                 ;Is this char inputted by user?
            je   reqcode_found         ;Yes
            cmp  ah,'*'                ;Have we reached the end of the table?
            je   reqcode_not_found     ;Yes, and we haven't found the req code
            add  si,3                  ;Point SI at next record in table
            jmp  reqcode_loop          ;Check next record

reqcode_found:
            mov  ax,word ptr [si+1]
            jmp  reqcode_exit

reqcode_not_found:
            showerr_msg 31             ;Unable to find request code error msg
            xor  ax,ax
            jmp  reqcode_exit

reqcode_exit:
            ret
get_reqcode endp

PAGE

;*******************************************************************
;*                                                                 *
;*  Procedure Name: CHECK_ARB                                      *
;*                                                                 *
;*  Description  : Checks to ensure that the ARB was found after   *
;*                 doing a call to one of the four modules in      *
;*                 the API/CS. The MACRO POPREGS saves the values  *
;*                 of the AX and DX registers in the variables     *
;*                 AX_REG and DX_REG so that these values are not  *
;*                 destroyed when the registers are restored. If   *
;*                 API/CS did not find the appropriate ARB ID in   *
;*                 the first four bytes of the ARB, due to its     *
;*                 having been passed either an invalid ARB address *
;*                 or an ARB which does not begin with a valid     *
;*                 ARB ID then the API/CS will set the AX and DX   *
;*                 registers to zero. If a valid ARB ID was passed *
;*                 the API/CS will return the AX:DX register pair  *
;*                 still pointing at the ARB that was passed.      *
;*                                                                 *
;*  Input : Registers AX and DX                                    *
;*          Variables AX_REG and DX_REG                            *
;*                                                                 *
;*  Output : The AL register is used to return the one character   *
;*           result. It will contain a 'Y' if the ARB was found,   *
;*           an 'N' if the ARB was not found or a 'U' if the       *
;*           AX and DX regs contain neither zeroes nor the address *
;*           of the ARB passed. The 'U' case should NEVER occur    *
;*          under any circumstances and the 'N' case should NEVER  *
;*          occur in this sample program.                          *
;*                                                                 *
;*******************************************************************
check_arb  proc near

           mov arb_found,'Y'             ;Assume ARB was found.

           cmp ax_reg,0                  ;Does AX = 0 ?
           je  check_dx                  ;Yes check DX
           jmp check_ax_good             ;No, then check to see
                                         ;if DX still points at ARB
check_dx:
           cmp dx_reg,0                  ;Is DX = 0 ?
           je  arb_not_found             ;Yes, API/CS could not find
                                         ;ARB.
           jmp check_ax_good             ;No, then check to see if AX
                                         ;still points at ARB
check_ax_good:
           cmp ax,ax_reg                 ;Does AX point at ARB ?
           je  check_dx_good             ;Yes, check DX
           jmp axdx_not_restored         ;No, AX:DX do not point at ARB

page

check_dx_good:
           cmp dx,dx_reg                 ;Does DX point at ARB ?
           jne axdx_not_restored         ;No, it does not
           jmp check_done                ;Yes, AX:DX points at ARB

axdx_not_restored:
           mov arb_found,'U'             ;Indicate that whether or
           jmp check_done                ;not ARB was found is
                                         ;unknown. This should
                                         ;NEVER occur.
arb_not_found:
           mov arb_found,'N'             ;Indicate ARB was not found
           jmp check_done

check_done:
           mov al,arb_found              ;Put returned var in AL
           ret

check_arb  endp
```

APIUTIL.DSG

```
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.
DGROUP  GROUP   DATA,STACK

STACK   SEGMENT BYTE STACK 'STACK'
        DB      256 DUP('STACK   ')       ;2K STACK AREA
STKTOP  DW      1
STACK   ENDS

DATA    SEGMENT PARA PUBLIC 'DATA'
        ASSUME  DS:DGROUP


        CR_LF EQU WORD PTR 0A0DH       ;ASCII Code for Carriage Return/Line Feed
        CR    EQU BYTE PTR 13D         ;ASCII Code for Carriage Return
        ESC   EQU BYTE PTR 27D         ;ASCII Code for Escape Code
        F1    EQU BYTE PTR 59D         ;Scan Code for F1  key
        F2    EQU BYTE PTR 60D         ;Scan Code for F2  key
        F3    EQU BYTE PTR 61D         ;Scan Code for F3  key
        F4    EQU BYTE PTR 62D         ;Scan Code for F4  key
        F5    EQU BYTE PTR 63D         ;Scan Code for F5  key
        F6    EQU BYTE PTR 64D         ;Scan Code for F6  key
        F7    EQU BYTE PTR 65D         ;Scan Code for F7  key
        F8    EQU BYTE PTR 66D         ;Scan Code for F8  key
        F9    EQU BYTE PTR 67D         ;Scan Code for F9  key
        F10   EQU BYTE PTR 68D         ;Scan Code for F10 key
        F11   EQU BYTE PTR 84D         ;Scan code for F11 key
        F12   EQU BYTE PTR 85D         ;Scan code for F12 key

        PAGEUP EQU BYTE PTR 73d         ;Page up scan code
        PAGEDN EQU BYTE PTR 81d         ;Page down scan code



;**********************************************************************
; File names for I/O
;**********************************************************************
ARBfile    DB     'ARBFILE.BIN  '      ;Name of file containing binary
                                       ;image of ARB for SPCF

ARB_temp   DB     'ARBFILE.TMP  '      ;File name used by Get No Parse proc
                                       ;to temporarily store an ARB from
                                       ;the parse of the received NMVT.

Nmvtfile   DB     'NMVTFILE.BIN '      ;Name of file containing binary
                                       ;image of Alert NMVT to be sent
                                       ;to API/CS

;**********************************************************************
; Variables used by Read_Nmvt
;**********************************************************************
nmvthandle      DW 0H                  ;File handle for NMVTFILE
Read_Nmvt_Stat  DW 0                   ;Status indicator for READ_NMVT subrout
Filename_Ptr    DW 0                   ;Parameter for READ_NMVT
Readbuff_Ptr    DW 0                   ;Parameter for READ_NMVT
Readbuff_Size   DW 0                   ;Parameter for READ_NMVT
Filesize        DW 0                   ;Size of File returned by subrout
                                       ;READ_NMVT
```

```
;***********************************************************************
; Variables used by Write_File
;***********************************************************************
filehandle          DW 0                    ;File handle for FILE
Write_File_Stat     DW 0                    ;Status indicator for WRITE_FILE
Writebuff_entry_cnt equ 5                   ;Number of entries in
                                            ;Writebuff_Ptr_Tbl
Writebuff_Index     DW 0                    ;Index for Writebuff_Ptr_Tbl

Writebuff_Ptr_Tbl   DB (Writebuff_entry_cnt)*6 DUP (0)
                                            ;Table of entries for WRITE_FILE
                                            ;Each entry consists of a doubleword
                                            ;buffer address followed by a
                                            ;one word buffer size.
;**********************************
;* buffer used by Hexb2asc
;**********************************
hexb2asc_buff db '0000'



;***********************************************************************
; EZ-VU Variables used by utility routines
;***********************************************************************
PARM6D    DB    'SETMSG VAPI'
MSGNUM6   DB    '0000 NMVTFILE'
LNGTH6PD  DW    LNGTH6PD - PARM6D

PARM9D    DB    'SETMSG VAPI'
MSGNUM9   DB    '0000'
LNGTH9PD  DW    LNGTH9PD - PARM9D

PARM15D    DB    'SETMSG VAPI0010'
LNGTH15PD  DW    LNGTH15PD - PARM15D

PARM16D    DB    'SETMSG VAPI0011 MSGBUFFR'
LNGTH16PD  DW    LNGTH16PD - PARM16D

PAGE

;***********************************************************************
; Storage used in DELAY
;***********************************************************************
delay_time      dw 0                ;used in delay subrout
old_sec         db 0                ;used in delay subrout

;***********************************************************************
; Word used to pass active keyset to Set_Active_Keys
;***********************************************************************
Active_Keys     DW      0            ;Active key word - argument to
                                     ;       set_active_keys

;***********************************************************************
; Function Keys
;***********************************************************************
ZF01_PARM       DB      'ZF01 C'
ZF01LP          DW      $-ZF01_PARM
ZF01            DB      'Xxxx'
ZF01LV          DW      $-ZF01
```

```
ZF02_PARM       DB      'ZF02 C'
ZF02LP          DW      $-ZF02_PARM
ZF02            DB      'Xxxx'
ZF02LV          DW      $-ZF02

ZF03_PARM       DB      'ZF03 C'
ZF03LP          DW      $-ZF03_PARM
ZF03            DB      'Xxxx'
ZF03LV          DW      $-ZF03

ZF04_PARM       DB      'ZF04 C'
ZF04LP          DW      $-ZF04_PARM
ZF04            DB      'Xxxx'
ZF04LV          DW      $-ZF04

ZF05_PARM       DB      'ZF05 C'
ZF05LP          DW      $-ZF05_PARM
ZF05            DB      'Xxxx'
ZF05LV          DW      $-ZF05

ZF06_PARM       DB      'ZF06 C'
ZF06LP          DW      $-ZF06_PARM
ZF06            DB      'Xxxx'
ZF06LV          DW      $-ZF06

ZF07_PARM       DB      'ZF07 C'
ZF07LP          DW      $-ZF07_PARM
ZF07            DB      'Xxxx'
ZF07LV          DW      $-ZF07

ZF08_PARM       DB      'ZF08 C'
ZF08LP          DW      $-ZF08_PARM
ZF08            DB      'Xxxx'
ZF08LV          DW      $-ZF08

ZF09_PARM       DB      'ZF09 C'
ZF09LP          DW      $-ZF09_PARM
ZF09            DB      'Xxxx'
ZF09LV          DW      $-ZF09

ZF10_PARM       DB      'ZF10 C'
ZF10LP          DW      $-ZF10_PARM
ZF10            DB      'Xxxx'
ZF10LV          DW      $-ZF10

ZF11_PARM       DB      'ZF11 C'
ZF11LP          DW      $-ZF11_PARM
ZF11            DB      'Xxxx'
ZF11LV          DW      $-ZF11

ZF12_PARM       DB      'ZF12 C'
ZF12LP          DW      $-ZF12_PARM
ZF12            DB      'Xxxx'
ZF12LV          DW      $-ZF12
```

```
ZFKEY_DELETE    DB      'VDELETE ZF'
ZFKEY_TO_DELETE DW      '10'            ;Move key to be deleted here
ZFKEY_DEL_END   DB      ' A'
ZFKEY_DELETEL   DW      $-ZFKEY_DELETE

ZF01_A          EQU     '10'            ;Note byte reversal
ZF02_A          EQU     '20'            ;Note byte reversal
ZF03_A          EQU     '30'            ;Note byte reversal
ZF04_A          EQU     '40'            ;Note byte reversal
ZF05_A          EQU     '50'            ;Note byte reversal
ZF06_A          EQU     '60'            ;Note byte reversal
ZF07_A          EQU     '70'            ;Note byte reversal
ZF08_A          EQU     '80'            ;Note byte reversal
ZF09_A          EQU     '90'            ;Note byte reversal
ZF10_A          EQU     '01'            ;Note byte reversal
ZF11_A          EQU     '11'            ;Note byte reversal
ZF12_A          EQU     '21'            ;Note byte reversal

;*********************************************************************
; Following are used for dynamically altering text on panels associated
;       with non-F action keys
;*********************************************************************
Keyline_Parm    DB      'KEYLINE C'
Keylinelp       DW      $-Keyline_Parm
Keyline         DB      48 DUP (' ')
Keylinelv       DW      $-Keyline

keylineoff      DW      0

Enter_Text      DB      'Enter   '
ENTER_TLEN      EQU     $-Enter_Text

Escape_Text     DB      'Esc   '
ESCAPE_TLEN     EQU     $-Escape_Text

PgUp_Text       DB      'PgUp   '
PGUP_TLEN       EQU     $-PgUp_Text

PgDn_Text       DB      'PgDn   '
PGDN_TLEN       EQU     $-PgDn_Text


;*********************************************************************
; EXECPGM and EZVU variables
;*********************************************************************

Ezvu_rc        DW ?             ;EZ-VU Return Code
Ezvu_Call_Addr DW 0
Ezvu_Rc_Msg    DW CR_LF
               DB 'EZ-VU II Return Code = '
Ezvu_Rc_Asc    DB 'XXXXX (decimal) at hex offset '
Ezvu_Addr_Asc  DB 'XXXX into your Code Segment'
               DW CR_LF
               DW CR_LF
               DB 'Press Any Key to Continue or Esc to End Program ...'
               DW CR_LF
               DW CR_LF
               DB '$'
```

```
not_enough_mem_msg dw cr_lf
                   db 'Inadequate memory available to run subprogram.'
                   dw cr_lf
                   dw cr_lf
                   db 'press any key to continue...'
                   dw cr_lf
                   dw cr_lf
                   db '$'

PAGE

cant_run_pgm_msg dw cr_lf
                 db 'unable to run subprogram.'
                 dw cr_lf
                 db 'exec function return code = '
exec_rc_asc      db 'xxxxx (decimal).'
                 dw cr_lf
                 dw cr_lf
                 db 'press any key to continue...'
                 dw cr_lf
                 dw cr_lf
                 db '$'

ret2tester  dw cr_lf
            db "To return to the sample program enter the dos command 'exit'."
            dw cr_lf
            dw cr_lf
            db '$'

curs_top db 0                         ;cursor top line value and
curs_bot db 0                         ;cursor bottom line value
                                      ;used to set cursor size when
                                      ;exiting to a secondary command
                                      ;processor in subrout execpgm

execpgm1 db 'c:\command.com'          ;program name parameter used by
         db 0                         ;subrout execpgm to invoke a
                                      ;second command processor.

cmdline  db 1                         ;used by subrout execpgm
cmdbegin db ' '                       ;blank command line
cmdend   db 13d                       ;carriage return

execblk  equ $                        ;exec control block used
envaddr  dw  0                        ;by subrout execpgm
cmdaddr  dd  cmdline                 ;
fcb1addr dd  0                       ;
fcb2addr dd  0                       ;

page

by_10    dw    10d                    ;used by subrout hex2dec and
by_100   dw   100d                    ;subrout hex2decz to convert
                                      ;hex value to decimal/ascii

by_16    dw    16d                    ;used by subrout hex2asc and
                                      ;subrout asc2hex
```

```
;**********************************************************************
; Buffer definitions
;**********************************************************************
Nmvtbuff        EQU    $                    ;Buffer for NMVT storage
Nmvtlngth       DW     0                    ;
Nmvtblock       DB     NMVTBUFF_SIZE/8 DUP('NMVT    ')

Arbbuff         EQU    $                    ;Buffer for Arb storage
Arbblock        DB     NMVTBUFF_SIZE/8 DUP('ARB ARB ')

;**********************************************************************
; Make public symbols public
;**********************************************************************

PUBLIC Nmvtbuff
PUBLIC Nmvtlngth
PUBLIC Nmvtblock

PUBLIC Arbbuff
PUBLIC Arbblock

PUBLIC Nmvtfile
PUBLIC ARBfile
PUBLIC ARB_temp

PUBLIC Read_Nmvt_Stat
PUBLIC Filename_Ptr
PUBLIC Readbuff_Ptr
PUBLIC Readbuff_Size
PUBLIC Filesize
PUBLIC Write_File_Stat
PUBLIC Writebuff_Ptr_Tbl
PUBLIC Ezvu_rc
PUBLIC Ezvu_Call_Addr
PUBLIC Ezvu_Rc_Msg
PUBLIC Active_Keys

PUBLIC Ezvu_Rc_Asc
PUBLIC Ezvu_Addr_Asc

;PUBLIC Asc2ebc_Tbl
;PUBLIC Ebc2asc_Tbl

PUBLIC Keyline_parm
PUBLIC Keylinelp
PUBLIC Keyline
PUBLIC Keylinelv

DATA    ENDS
```

APIUTIL.EXR

```
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.
;
;***********************************************************************
;
; APIUTIL.EXR
;
; Include this file in any procedures using subroutines in APIUTIL.ASM
;
;***********************************************************************

;***********************************************************************
; Statements below allow other programs to access utility variables
;***********************************************************************
extrn Nmvtfile:byte
extrn ARBfile:byte
extrn ARB_temp:byte

extrn Nmvtbuff:byte
extrn Nmvtlngth:word
extrn Nmvtblock:byte

extrn Arbbuff:byte
extrn Arbblock:byte

extrn Read_Nmvt_Stat:word
extrn Filename_Ptr:word
extrn Readbuff_Ptr:word
extrn Readbuff_Size:word
extrn Filesize:word
extrn Write_File_Stat:word
extrn Writebuff_Ptr_Tbl:byte
extrn Ezvu_rc:word
extrn Ezvu_Call_Addr:word
extrn Ezvu_Rc_Msg:word
extrn Active_Keys:word

extrn Ezvu_Rc_Asc:byte
extrn Ezvu_Addr_Asc:byte

extrn Keyline_parm:byte
extrn Keylinelp:word
extrn Keyline:byte
extrn Keylinelv:word

;***********************************************************************
; Define the utility routines available in APIUTIL.ASM
;***********************************************************************
```

```
extrn Check_Ezvu_Rc:near
extrn Execpgm:near
extrn Delay:near
extrn Clrscr:near
extrn Decasc2bin:near
extrn Hex2decz:near
extrn Hex2dec:near
extrn Hex2asc:near
extrn Asc2hex:near
extrn show_errmsg:near
extrn Read_Nmvt:near
extrn Write_File:near
extrn Hexb2asc:near
extrn Set_Active_Keys:near
```

APIUTIL.ASM

```
; (CTRL-OH) IBM PC PRINTER CONDENSED MODE
PAGE  ,132
TITLE API Sample Program Utility Routines (C) Copyright IBM Corp. 1986,1987
;        SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.


        INCLUDE  APIMAIN.DEF            ;Shared constants
        INCLUDE  APIUTIL.DSG            ;Data Segment and references
        INCLUDE  APIMAIN.EXR            ;Shared variables & procedures
         IF1
           INCLUDE  APIMAIN.MAC         ;Macros
         ELSE
          %OUT Starting second pass ...
         ENDIF

PAGE

PGROUP  GROUP   CSEG

CSEG    SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:PGROUP,DS:DGROUP,ES:DGROUP,SS:NOTHING

        EXTRN ISPASM:FAR              ;EZ-VU II Display functions
        EXTRN ISPASMV:FAR             ;EZ-VU II Variable definitions
        EXTRN ISPASMVA:FAR            ;EZ-VU II Variable definitions
;*********************************************************************
; This code is linked in rather than being included with the main API
;       program, so some things need to be shared
;       PUBLIC declarations for procedures immediately precede the
;       procedure names
;*********************************************************************
```

```
PAGE
;*********************************************************************
;*
;* Description : Writes one or more buffers to a specified file
;*
;* Input  : Filename_Ptr will contain the address of the file name
;*          to which to write.
;*          Writebuff_Ptr_Tbl will contain the address(es) and size(s)
;*          of the buffer(s) to write to the file. Each entry in this
;*          table is 6 bytes long, consisting of a double word buffer
;*          address followed by a one word buffer size. The end of the
;*          table is marked by an entry whose size entry is zero or by
;*          physical end of the table which is Writebuff_entry_cnt
;*          entries long.
;*
;* Output : The buffer will be written to the designated file from the
;*          designated buffer. Write_File_Stat will be set to zero if the
;*          WRITE is performed successfully, otherwise it will be set to FFH
;*
;*********************************************************************
PUBLIC Write_File
Write_File proc near
          PUSHREGS
          mov  Writebuff_Index,0          ;Init Index to beg of
                                          ; Writebuff_Ptr_Tbl
          mov  Write_File_Stat,0          ;Init status flag to successful
          mov  di,Filename_Ptr            ;Point DI at filename

          call delimit_fn                 ; Zero delimit filename

          mov  dx,di                      ;Point DX at filename

                                          ;Create the file
          xor  cx,cx                      ;  File attribute - normal
          mov  ah,3ch                     ;  Open output file function code
          int  21h                        ;  Call DOS

          mov  Filehandle,ax              ;Save File Handle

          jnc  good_file_open             ;Test for Open Error
          jmp  file_open_error

good_file_open:
          mov  bx,Filehandle              ;Put  File Handle in BX

          mov  di,Writebuff_Index         ;Point di at current entry
                                          ;in Writebuff_Ptr_Tbl

          cmp  Writebuff_Index,Writebuff_entry_cnt*6
                                          ;If physical end of tbl has been
          ja   close_the_wrt_file         ;reached, exit this loop.
                                          ;Set up bytes to write
          mov  cx,word ptr Writebuff_Ptr_Tbl[di+4]
          jcxz close_the_wrt_file         ;If zero end of tbl has been
                                          ;reached so exit loop.
          push ds                         ;save ds
```

```
                                    ;Point ds:dx at write buffer
          lds  dx,dword ptr Writebuff_Ptr_Tbl[di]

          mov  ah,40h               ;Write file function code
          int  21h                  ;Call DOS
          pop  ds                   ;restore ds

          jc   file_write_error     ;Was there an error ?

          add  Writebuff_Index,6    ;Bump Writebuff_Index to
                                    ;point at next entry in tbl
          jmp  good_file_write      ;Test for Write Error

good_file_write:
          cmp  ax,cx                ;Test for End of File
          je   good_file_open       ;If not EOF, process next tbl entry
          jmp  file_write_error     ;Else, show error msg and exit

close_the_wrt_file:
          mov  bx,Filehandle        ;Put file handle to Close in BX
          mov  ah,3eh               ;Function Code for Close File
          int  21h                  ;Call DOS
          jnc  good_file_close_w    ;Test for Close Error
          jmp  file_close_error

good_file_close_w:
          jmp  write_file_exit      ;Exit subrout

file_open_error:
          mov  ioretcod,ax          ;set error msg ret code variable
          mov  ax,209d              ;msg VAPI0209 - file create error
          jmp  write_file_error_exit

file_write_error:
          mov  ioretcod,ax          ;set error msg ret code variable
          mov  ax,210d              ;msg VAPI0210 - FILE WRITE error
          jmp  write_file_error_exit

file_close_error:
          mov  ioretcod,ax          ;set error msg ret code variable
          mov  ax,7                 ;msg VAPI0007 - FILE close error
          jmp  write_file_error_exit

write_file_error_exit:
          mov  Write_File_Stat,0ffh ;Indicate failure to caller
          lea  di,msgnum6           ;Build SETMSG string
          call Hex2decz
          DMPC ISPASM,<LNGTH6PD,PARM6D,EZVU_RC>  ;Display error msg
          jmp write_file_exit        ;Branch to subrout exit


write_file_exit:
          POPREGS                   ;Restore all regs
          ret
Write_File endp


delimit_fn proc near
```

```
;**************************************************************
;*
;* Description : Puts a binary zero char at the end of the
;*               of the filename pointed at di.
;*
;* Input:  di points at file name to be zero delimited.
;*
;* Output: Filename is delimited with a binary zero terminator.
;*
;**************************************************************
          push di
          push ax

fname_loop:                         ;Search for the end of the file
          inc  di                   ;name so you can put a zero
          mov  al,[di]              ;delimiter after it.
          cmp  al,' '               ;Is char a blank ?
          je   eofn_fnd             ;Yes, End of name found
          cmp  al,0                 ;Is char a binary zero ?
          jne  fname_loop           ;No, look at next char

eofn_fnd:
          xor  al,al                ;Put Zero delimiter at end
          mov  [di],al              ;of file name.

          pop  ax
          pop  di

          ret
delimit_fn endp

PAGE
;*********************************************************************
;*
;* Description : Reads a binary NMVT or ARB image from a specified file into
;*               a specified buffer.
;*
;* Input  : Filename_Ptr will contain the address of the file name
;*          from which to read the NMVT or ARB.
;*          Readbuff_Ptr will contain the address of the buffer into
;*          which to read the NMVT or ARB.
;*          Readbuff_Size will contain the size in bytes of the buffer
;*          into which to read the NMVT or ARB.
;*
;* Output : The NMVT or ARB will be read from the designated file into the
;*          designated buffer. Read_Nmvt_Stat will be set to zero if the
;*          read is performed successfully, otherwise it will be set to FFH
;*
;*
;*********************************************************************
PUBLIC Read_Nmvt
Read_Nmvt  proc near
          PUSHREGS
          mov  Read_Nmvt_Stat,0     ;Init status flag to successful
          mov  di,Filename_Ptr      ;Point DI at filename

          call delimit_fn           ; Zero delimit filename

          mov  dx,di                ;Point DX at filename
```

```
                                           ;Open the NMVT file
           xor  al,al                      ;  Access code 0 - read only,in AL
           mov  ah,3dh                     ;  Open file function code
           int  21h                        ;  Call DOS

           jnc  good_nmvt_open             ;  Test for Open Error
           jmp  nmvt_open_error

good_nmvt_open:
           mov  nmvthandle,ax              ;Save File Handle

           mov  bx,nmvthandle              ;Put file handle in BX
           xor  cx,cx                      ;Determine size of
           xor  dx,dx                      ;file by moving zero
           mov  al,2                       ;bytes past the EOF.
           mov  ah,42h                     ;Move file pointer function
           int  21h                        ;  Call DOS

           jnc  move_ptr_good1             ;Test for Error
           jmp  move_ptr_error             ;

move_ptr_good1:
           mov  Filesize,ax                ;Save file size which was
                                           ;returned in DX:AX.
           cmp  dx,0                       ;Is file size > 64K error
           je   filesize_lt_64k            ;No
           jmp  nmvt_too_big               ;Yes, error

filesize_lt_64k:
           mov  bx,nmvthandle              ;Put file handle in BX
           xor  cx,cx                      ;Reposit file ptr to
           xor  dx,dx                      ;beginning of file.
           mov  al,0                       ;
           mov  ah,42h                     ;Move file pointer function
           int  21h                        ;  Call DOS

           jnc  move_ptr_good2             ;Test for Error
           jmp  move_ptr_error             ;

move_ptr_good2:
           mov  cx,Filesize                ;Set CX to size of file
           cmp  cx,Readbuff_Size           ;Will NMVT fit in the target buffer?
           jbe  nmvt_will_fit              ;Yes
           jmp  nmvt_too_big               ;No

nmvt_will_fit:
           mov  bx,nmvthandle              ;Put file handle in BX
           mov  dx,Readbuff_Ptr            ;Point DX at read buffer
           mov  ah,3fh                     ;Read file function code
           int  21h                        ;Call DOS
           jnc  good_nmvt_read             ;Test for Read Error
           jmp  nmvt_read_error

good_nmvt_read:
           cmp  ax,cx                      ;Test for End of File
           je   not_past_eof
           jmp  nmvt_read_error
```

```
PAGE

not_past_eof:
          mov  bx,nmvthandle            ;Put file handle to Close in BX
          mov  ah,3eh                   ;Function Code for Close File
          int  21h                      ;Call DOS
          jnc  good_nmvt_close          ;Test for Close Error
          jmp  nmvt_close_error

good_nmvt_close:
          jmp  read_nmvt_exit           ;Exit subrout

nmvt_too_big:
          mov  ax,30                    ;msg VAPI0030 - NMVT too big
          jmp  read_nmvt_error_exit

nmvt_open_error:
          mov  ioretcod,ax              ;set error msg ret code variable
          mov  ax,5                     ;msg VAPI0005 - NMVT open error
          jmp  read_nmvt_error_exit

nmvt_read_error:
          mov  ioretcod,ax              ;set error msg ret code variable
          mov  ax,6                     ;msg VAPI0006 - NMVT read error
          jmp  read_nmvt_error_exit

nmvt_close_error:
          mov  ioretcod,ax              ;set error msg ret code variable
          mov  ax,7                     ;msg VAPI0007 - NMVT close error
          jmp  read_nmvt_error_exit

move_ptr_error:
          mov  ioretcod,ax              ;set error msg ret code variable
          mov  ax,32                    ;msg VAPI0032 - Move ptr error
          jmp  read_nmvt_error_exit

read_nmvt_error_exit:
          mov  Read_Nmvt_Stat,0ffh      ;Indicate failure to caller
          lea  di,msgnum6               ;Build SETMSG string
          call Hex2decz
          DMPC ISPASM,<LNGTH6PD,PARM6D,EZVU_RC>  ;Display error msg
          jmp read_nmvt_exit            ;Branch to subrout exit


read_nmvt_exit:
          POPREGS                       ;Restore all regs
          ret
Read_Nmvt  endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: SHOW_ERRMSG                                    *
;*                                                                 *
;*  Description  : Inserts the message number contained in AX      *
;*                 into a SETMSG string and uses it to call        *
;*                 EZVU II to display the message.                 *
;*                                                                    *
;*  Input : AX contains the message number.                           *
;*                                                                    *
;*  Output : The requested message is displayed if it exists.         *
;*                                                                    *
;**********************************************************************
PUBLIC show_errmsg
show_errmsg  proc near
             push di                   ;Save DI
             lea  di,msgnum9           ;Insert message number into string
             call Hex2decz            ;
             pop  di                   ;Restore DI
             DMPC ISPASM,<LNGTH9PD,PARM9D,EZVU_RC>   ;Display error msg
             ret
show_errmsg  endp

PAGE
;**********************************************************************
;*                                                                    *
;*  Procedure Name: Asc2hex                                           *
;*                                                                    *
;*  Description  : Converts the Hex/ASCII string pointed at by DI    *
;*                 and whose length is contained in CX to a Hex       *
;*                 value that is returned in AX. If non-Hex chars     *
;*                 are found in the string CX is set to -1 on         *
;*                 return.                                            *
;*                                                                    *
;*  Input : Register DI points at the Hex/ASCII string to be          *
;*          converted.                                                *
;*          Register CX contains the length of the string (max 4).   *
;*                                                                    *
;*  Output : Register AX is used to return the result of the          *
;*           conversion. If non-Hex chars were found in the input     *
;*           string CX will be set to -1 on return, otherwise         *
;*           CX will be set to zero.                                  *
;*                                                                    *
;**********************************************************************
PUBLIC Asc2hex
Asc2hex proc near
        push    dx                   ;Save regs
        push    di

        xor     ax,ax                ;Clear AX

asc2hex_loop:
        mul     by_16                ;Shift AX left one Hex digit
        xor     dx,dx                ;Clear DX
        mov     dl,[di]              ;Put next char of ASCII/HEX string
                                     ;in DL
        cmp     dl,' '
        jne     not_a_blank_in_rc    ;Is this char a blank ?

        mov     dl,'0'               ;Yes, change it to a '0'
        mov     [di],dl

not_a_blank_in_rc:
        sub     dl,48d               ;Subtract ASCII code for '0' from
                                     ;DL to convert to number 0 - 22
        cmp     dl,9                ;If number > 9 then subtract 7
        jle     not_athruf          ;from number to bridge gap from
        sub     dl,7                ;ASCII '9' to ASCII 'A'
                                    ;Number should now be converted to range of 0-15

PAGE

not_athruf:
        cmp     dx,15               ;Number should now be converted to
                                    ;range of 0-15.
        ja      badcode_done        ;Is number in range ?

        add     ax,dx               ;Yes number is in range.
                                    ;Add number to AX

        inc     di                  ;Bump DI to char in string

        loop    asc2hex_loop        ;Loop until all chars processed.

        jmp     asc2hex_done        ;Exit subrout

badcode_done:                       ;Number is out of range.
         mov    cx,-1
         jmp    asc2hex_done

asc2hex_done:
         pop     di                 ;Restore regs
         pop     dx
         ret


Asc2hex  endp

PAGE

;*********************************************************************
;*                                                                   *
;*  Procedure Name: Hex2asc                                          *
;*                                                                   *
;*  Description  : Converts the value contained in AX to a four     *
;*                 character Hex/ASCII string pointed at by DI.      *
;*                                                                   *
;*  Input : Register DI points at a 4 char string buffer to be      *
;*          used as the target buffer for the conversion.            *
;*          Register AX contains value to be converted.              *
;*                                                                   *
;*  Output : The target buffer pointed at by DI will contain        *
;*           a four char Hex-ASCII string that is the                *
;*           representation of the value passed in AX.               *
;*                                                                   *
;*********************************************************************

PUBLIC Hex2asc
Hex2asc  proc near
         push    ax                       ;save regs
         push    dx
         push    di

         mov     byte ptr [di]  ,'0'      ;init output string to
        mov     byte ptr [di+1],'0'      ;all zeroes
        mov     byte ptr [di+2],'0'
        mov     byte ptr [di+3],'0'
        add     di,3

hexloop:
        xor     dx,dx
        div     by_16                    ;divide ax by 16
                                         ;quotient in ax
                                         ;remainder in dx

        add     dx,30h                   ;add ascii zero to remainder
                                         ;to convert char to ascii

        cmp     dx,39h                   ;if char is > ascii '9'
        jle     not_a_f
        add     dx,7                     ;add 7 to bridge gap between
                                         ;ascii '9' and ascii 'A'

not_a_f:

        mov     byte ptr [di],dl         ;store ascii char in string
        dec     di

        cmp     ax,0                     ;are we through, y/n ?
        jz      hexdone                  ;yes
        jmp     hexloop                  ;no, convert next char

page

hexdone:
        pop     di                       ;restore regs
        pop     dx
        pop     ax

        ret


Hex2asc endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: Hex2dec                                        *
;*                                                                 *
;*  Description  : Converts the value contained in AX to a five    *
;*                 character Decimal/ASCII string padded with      *
;*                 leading blanks pointed at by DI.                *
;*                                                                 *
;*  Input : Register DI points at a 5 char string buffer to be     *
;*          used as the target buffer for the conversion.          *
;*          Register AX contains value to be converted.            *
;*                                                                 *
;*  Output : The target buffer pointed at by DI will contain       *
;*           a five char Dec-ASCII string with leading blanks      *
;*           that is the representation of the value passed in AX. *
;*                                                                 *
;*  Restriction : This routine cannot convert negative numbers.    *
;*                If a negative number is passed to it it will     *
;*               set the target string to 'NEGTV' and return.      *
;*                                                                 *
;*******************************************************************

PUBLIC Hex2dec
Hex2dec  proc near
         push    ax                       ;save regs
         push    dx
         push    di

         mov     byte ptr [di]  ,' '      ;init output string to
         mov     byte ptr [di+1],' '      ;all blanks
         mov     byte ptr [di+2],' '
         mov     byte ptr [di+3],' '
         mov     byte ptr [di+4],' '

         mov     dx,ax                    ;check to see if number
         and     dx,8000h                 ;is negative
         cmp     dx,0
         jne     num_is_neg               ;yes, it is negative
         jmp     num_is_pos               ;no, it is positive

num_is_neg:
         mov     byte ptr [di]  ,'n'      ;set output string equal
         mov     byte ptr [di+1],'e'      ;to 'negtv' and exit
         mov     byte ptr [di+2],'g'
         mov     byte ptr [di+3],'t'
         mov     byte ptr [di+4],'v'
         jmp     decdone

num_is_pos:
         add     di,4                     ;point di at last char in
                                          ;output string
page

decloop:
         xor     dx,dx                    ;clear dx
         div     by_10                    ;divide ax by 10
                                          ;quotient in ax
                                          ;remainder in dx

         add     dx,30h                   ;add ascii zero to remainder
         mov     byte ptr [di],dl         ;store ascii char in string
         dec     di

         cmp     ax,0                     ;are we through yet, y/n?
         jz      decdone                  ;yes
         jmp     decloop                  ;no, convert next char
decdone:
         pop     di                       ;restore regs
         pop     dx
         pop     ax

         ret

Hex2dec  endp

PAGE
;*******************************************************************
```

```
;*                                                                  *
;*  Procedure Name: Hex2decz                                        *
;*                                                                  *
;*  Description  : Converts the value contained in AX to a four     *
;*                 character Decimal/ASCII string padded with       *
;*                 leading zeroes pointed at by DI.                 *
;*                                                                  *
;*  Input : Register DI points at a 4 char string buffer to be      *
;*          used as the target buffer for the conversion.           *
;*          Register AX contains value to be converted.             *
;*                                                                  *
;*  Output : The target buffer pointed at by DI will contain        *
;*           a four char Dec-ASCII string with leading zeroes       *
;*           that is the representation of the value passed in AX. *
;*                                                                  *
;*  Restriction : This routine cannot convert negative numbers.     *
;*                If a negative number is passed to it it will      *
;*                set the target string to 'NEGT' and return.       *
;*                                                                  *
;*******************************************************************

PUBLIC Hex2decz
Hex2decz proc near
        push    ax                      ;Save regs
        push    dx
        push    di

        mov     byte ptr [di]  ,'0'     ;Init output string
        mov     byte ptr [di+1],'0'     ;to all zeroes
        mov     byte ptr [di+2],'0'
        mov     byte ptr [di+3],'0'

        mov     dx,ax
        and     dx,8000h
        cmp     dx,0                    ;Is number negative ?
        jne     num_is_negz             ;Yes
        jmp     num_is_posz             ;No

num_is_negz:
        mov     byte ptr [di]  ,'n'     ;Set target string equal to
        mov     byte ptr [di+1],'e'     ;'NEGT' and return
        mov     byte ptr [di+2],'g'
        mov     byte ptr [di+3],'t'
        jmp     decdonez

num_is_posz:
        add     di,3                    ;Point DI at last char in string

page

decloopz:
        xor     dx,dx
        div     by_10                   ;DIVIDE AX BY 10
                                        ;QUOTIENT IN AX
                                        ;REMAINDER IN DX

        add     dx,30h                  ;ADD ASCII ZERO TO REMAINDER
        mov     byte ptr [di],dl        ;STORE ASCII CHAR IN STRING
        dec     di
        cmp     ax,0                    ;Are we through yet ?
        jz      decdonez                ;Yes
        jmp     decloopz                ;No
decdonez:
        pop     di                      ;Restore regs
        pop     dx
        pop     ax

        ret


Hex2decz endp

PAGE

;*********************************************************************
;*                                                                   *
;*  Procedure Name: Decasc2bin                                       *
;*                                                                   *
;*  Description  : Converts the Dec/ASCII string of length 3         *
;*                 pointed at by SI to a binary value which is       *
;*                 returned.in CX.                                   *
;*                 If non-decimal numeric chars are found in the     *
;*                 string CX is set to -1 on return.                 *
;*                                                                   *
;*  Input : Register SI points at the Dec/ASCII string to be         *
;*          converted.                                               *
;*                                                                   *
;*  Output : Register CX is used to return the result of the         *
;*           conversion. If non-Dec chars were found in the input    *
;*           string CX will be set to -1 on return.                  *
;*                                                                   *
;*********************************************************************

PUBLIC Decasc2bin
Decasc2bin proc near
          push ax                       ;Save AX
;*
;* This loop will verify that the chars in the string are valid chars
;*

          mov  cx,3                     ;Set CX to length of string
                                        ;for use as loop counter

check_num:
          cmp  byte ptr [si],' '        ;Replace blanks with zeroes
          jne  not_a_blank_in_num
          mov  byte ptr [si],'0'
not_a_blank_in_num:
          cmp  byte ptr [si],'0'        ;Is char < '0' ?
          jae  char_not_too_low         ;Yes, invalid char
          jmp  bad_dec_char             ;Exit routine
char_not_too_low:
          cmp  byte ptr [si],'9'        ;Is char > '9' ?
          jbe  char_not_too_high        ;Yes, invalid char
          jmp  bad_dec_char             ;Exit routine
```

```
char_not_too_high:
          dec  cx                        ;Decrement loop counter
          inc  si                        ;Bump ptr to next char in string
          jcxz convert_num               ;Finished checking ?
          jmp  check_num                 ;No

page

convert_num:
          sub  si,3                      ;Point SI at leftmost char in
                                         ;input string
          xor  cx,cx                     ;Clear CX

          mov  cl,[si+2]                 ;Put One's char in CL
          sub  cx,48d                    ;Convert from ASCII to 0-9

          xor  ax,ax                     ;Clear AX
          mov  al,[si+1]                 ;Put Ten's char in AL
          sub  ax,48d                    ;Convert from ASCII to 0-9
          mul  by_10                     ;Mult by 10
          add  cx,ax                     ;Add Ten's to One's

          xor  ax,ax                     ;Clear AX
          mov  al,[si]                   ;Put Hundred's char in AL
          sub  ax,48d                    ;Convert from ASCII to 0-9
          mul  by_100                    ;Mult by 100
          add  cx,ax                     ;Add Hundred's to Ten's & One's

          cmp  cx,253                    ;Result > 253 ?
          ja   bad_msg_length            ;Yes, exceeds max msg length
          cmp  cx,0                      ;Result <= 0
          je   bad_msg_length            ;Yes, result less than min msg length
          jmp  decasc2bin_exit           ;No, valid value

bad_msg_length:
;*
;* Display error message indicating that length is not within acceptable bounds
;*
          DMPC ISPASM,<LNGTH15PD,PARM15D,EZVU_RC>
          jmp  decasc2bin_exit

bad_dec_char:
;*
;* Display error message indicating that non-numeric chars were found in string
;*
          DMPC ISPASM,<LNGTH16PD,PARM16D,EZVU_RC>
          mov  cx,-1                     ;Indicate error in conversion
          jmp  decasc2bin_exit           ;Exit routine

decasc2bin_exit:
          pop  ax                        ;Restore AX
          ret

Decasc2bin endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: Clrscr                                         *
;*                                                                  *
;*  Description  : Clears the screen and sets the variables used    *
;*                 for setting cursor size.                         *
;*                                                                  *
;*  Input : None                                                    *
;*                                                                  *
;*  Output : CURS_TOP and CURS_BOT are set to the appropriate       *
;*           values for the type of monitor on which the program    *
;*           is being run.                                          *
;*                                                                  *
;********************************************************************

PUBLIC Clrscr
Clrscr   proc    near
         push    ax                       ;SAVE AX
         mov     ah,0fh                   ;GET CURRENT VIDEO MODE
         int     10h                      ;MODE IS RETURNED IN AL

         mov     curs_top,6               ;Assume color monitor and
         mov     curs_bot,7               ;set vals for cursor top/bot
         cmp     al,7                     ;Is it mono?
         jne     not_monochrome           ;No leave top/bot as they are.
         mov     curs_top,12
         mov     curs_bot,13

not_monochrome:
         mov     ah,0                     ;RESET CURRENT VIDEO MODE
         int     10h                      ;TO WHAT IT ALREADY IS WHICH
                                          ;WILL CLEAR THE SCREEN

         pop     ax                       ;RESTORE AX
         ret

Clrscr   endp

PAGE
;********************************************************************
;*                                                                  *
;*  Procedure Name: Delay                                           *
;*                                                                  *
;*  Description  : Does a processor independent delay for the      *
;*                 number of seconds passed in AX.                  *
;*                                                                  *
;*  Input : AX contains the number of seconds to delay.             *
;*                                                                  *
;*  Output : None                                                   *
;*                                                                  *
;********************************************************************
PUBLIC Delay
Delay    proc   near
         PUSHREGS                         ;Save regs
         mov delay_time,ax                ;Save delay time

do_delay:
         mov ah,2ch                       ;Get Time delay begun
         int 21h

delay_loop:
         mov old_sec,dh
         cmp delay_time,0                ;Enough time elapsed yet ?
         je  end_delay_loop              ;Yes

get_time_loop:
         mov ah,2ch                      ;Get Time
         int 21h
         cmp old_sec,dh                  ;Have seconds changed ?
         je  get_time_loop               ;No, keep looping

         dec delay_time                  ;Another second has passed.
         jmp delay_loop

end_delay_loop:
         POPREGS                         ;Restore regs
         ret
Delay    endp

PAGE
;*******************************************************************
;*                                                                 *
;*  Procedure Name: Execpgm                                        *
;*                                                                 *
;*  Description  : Uses the DOS EXEC function to invoke a second   *
;*                 copy of the command processor.                  *
;*                                                                 *
;*  Input : None                                                   *
;*                                                                 *
;*  Output : Error messages if the EXEC fails.                     *
;*                                                                 *
;*******************************************************************
PUBLIC Execpgm
Execpgm  proc near
         jmp past_regsave_area

ss_save  dw ?            ;Save area for stack pointers for restoration
sp_save  dw ?            ;after an EXEC (4BH) function call to DOS

past_regsave_area:
         PUSHREGS                        ;Save all registers
         call Clrscr                     ;Clear the screen

         lea  dx,ret2tester              ;Display the Return to Tester message.
         mov  ah,9
         int  21h

         mov  ah,1                       ;Set cursor size for
         mov  ch,curs_top                ;exit to secondary DOS
         mov  cl,curs_bot
         int  10h

         mov ah,62h                      ;Get Addr of beginning of pgm
         int 21h

         mov  es,bx                      ;Request that pgm size be
         mov  bx,4096d                   ;limited to 4096 paragraphs
         mov  ah,4ah                     ;which is  64K
         int  21h

         mov  ax,es:2ch                  ;Set up environment ptr
```

```
        mov  envaddr,ax                 ;for subprogram load.
        lea  dx,Execpgm1
        lea  bx,execblk
        mov  ax,ds
        mov  es,ax

page

        push ds                         ;Save Data Seg

        mov  ax,ss                      ;On return from EXEC function
        mov  ss_save,ax                 ;all regs may be clobbered
        mov  ax,sp                      ;including SS and SP. One
        mov  sp_save,ax                 ;place to save these pointers
                                        ;is in the Code Segment.


        mov  al,0                       ;Indicate subpgm to be executed
        mov  ah,4bh                     ;Exec function code
        int  21h                        ;Run sub program
        mov  cx,ax                      ;Save return code in CX

        mov  ax,sp_save                 ;Put saved stack pointer values in
        mov  bx,ss_save                 ;AX and BX in order to minimize number
                                        ;of instruction to be performed with
                                        ;interrupts disabled.

        cli                        ;Disable interrupts while switching stacks
        mov  sp,ax                 ;Restore stack offset ptr
        mov  ss,bx                 ;Restore stack segment ptr
        sti                        ;Re-enable interuppts

        pop  ds                         ;Restore Data Seg

        cmp  cx,8                       ;Did it have enough memory?
        jne  enough_mem                 ;Yes
        lea  dx,not_enough_mem_msg      ;No, Display the message
        mov  ah,9
        int  21h

        mov  ah,0                       ;Wait for a keystroke
        int  16h                        ;Call the BIOS
        jmp  pgm_ran_good

enough_mem:
        cmp  cx,1                       ;Invalid function number
        je   exec_failed
        cmp  cx,2                       ;File not found
        je   exec_failed
        cmp  cx,5                       ;Access denied
        je   exec_failed
        cmp  cx,10                      ;Invalid environment
        je   exec_failed
        cmp  cx,11                      ;Invalid format
        jne  pgm_ran_good
page

exec_failed:
        mov  ax,cx                      ;Insert Return code into
        lea  di,exec_rc_asc            ;message.
        call Hex2dec
        lea  dx,cant_run_pgm_msg       ;Display the message
        mov  ah,9
        int  21h

        mov  ah,0                      ;Wait for a keystroke
        int  16h                       ;Call the BIOS

pgm_ran_good:
        call Clrscr                    ;Clear the screen

        POPREGS                        ;Restore all regs
        ret
Execpgm  endp

PAGE
;*********************************************************************
;*                                                                   *
;*  Procedure Name: Check_Ezvu_Rc                                    *
;*                                                                   *
;*  Description  : This procedure is called after all EZVU function *
;*                 calls made using the DMPC MACRO. If the EZVU      *
;*                 return code EZVU_RC is zero, no action is taken. *
;*                 If EZVU_RC is non-zero, the screen is cleared     *
;*                 and a message is displayed showing the return     *
;*                 code returned by EZVU and the offset into the     *
;*                 code segment of the call that received the        *
;*                 non-zero return code.                             *
;*                                                                   *
;*  Input : EZVU_RC contains the return code from the last call to  *
;*          EZVU.                                                    *
;*          EZVU_CALL_ADDR contains the offset into the code         *
;*          of the last call to EZ-VU.                               *
;*                                                                   *
;*  Output : Error messages whenever EZVU_RC is non-zero.            *
;*                                                                   *
;*  Note : There are instances in which a non-zero return code from *
;*         EZ-VU does not necessarily signal an error condition.     *
;*         For such instances, the action taken by this procedure    *
;*         may not be appopriate. There are , however no such        *
;*         instances in this sample program.                         *
;*                                                                   *
;*********************************************************************
PUBLIC Check_Ezvu_Rc
Check_Ezvu_Rc  proc near
               PUSHREGS                     ;Store regs

               cmp  ezvu_rc,0               ;Is return code zero ?
               jne  show_ezvu_rc_errmsg     ;No, error has occurred
               jmp  check_ezvu_rc_exit      ;Yes, call was successful

show_ezvu_rc_errmsg:
               mov  ax,ezvu_rc              ;Insert Return Code into
               lea  di,ezvu_rc_asc          ;message
               call Hex2dec
               mov  ax,ezvu_call_addr       ;Insert Call Address into
               lea  di,ezvu_addr_asc        ;message
               call Hex2asc
               call Clrscr                    ;Clear the screen
               lea  dx,ezvu_rc_msg            ;Display the message
               mov  ah,9
               int  21h

               mov  ah,0                       ;Wait for a keystroke
               int  16h                        ;Call the BIOS
               cmp  al,esc                     ;Was it the ESC key ?
               jne  check_ezvu_rc_exit
               mov  ax,ezvu_rc                 ;Set ErrorLevel for exit
               mov  ah,4ch                    ;'RETURN TO DOS' FUNCTION CALL
               int  21h                        ;RETURN TO DOS

check_ezvu_rc_exit:
               POPREGS                         ;Restore regs
               ret
Check_Ezvu_Rc  endp


;*******************************************************************
;*                                                                 *
;* Set_Active_Keys                                                 *
;*                                                                 *
;* Input:                                                          *
;*      Active_Keys     - word with flags set indicating which keys *
;*                              are valid                          *
;*                                                                 *
;* Output:                                                         *
;*      ZFxx            - Set to END if valid, 0h'ND' if invalid   *
;*      ZENTxx          - Set to scan codes of alternate enter keys *
;*                                                                 *
;*******************************************************************
PUBLIC Set_Active_Keys
Set_Active_Keys proc    near
        PUSHREGS

;First, set all F keys active so we can just turn them off later

;***********************************************************************
;Define the function keys
;***********************************************************************
        DMPC_NC ISPASMV,<ZF01LP,ZF01_PARM,EZVU_RC,ZF01,ZF01LV>
        DMPC_NC ISPASMV,<ZF02LP,ZF02_PARM,EZVU_RC,ZF02,ZF02LV>
        DMPC_NC ISPASMV,<ZF03LP,ZF03_PARM,EZVU_RC,ZF03,ZF03LV>
        DMPC_NC ISPASMV,<ZF04LP,ZF04_PARM,EZVU_RC,ZF04,ZF04LV>
        DMPC_NC ISPASMV,<ZF05LP,ZF05_PARM,EZVU_RC,ZF05,ZF05LV>
        DMPC_NC ISPASMV,<ZF06LP,ZF06_PARM,EZVU_RC,ZF06,ZF06LV>
        DMPC_NC ISPASMV,<ZF07LP,ZF07_PARM,EZVU_RC,ZF07,ZF07LV>
        DMPC_NC ISPASMV,<ZF08LP,ZF08_PARM,EZVU_RC,ZF08,ZF08LV>
        DMPC_NC ISPASMV,<ZF09LP,ZF09_PARM,EZVU_RC,ZF09,ZF09LV>
        DMPC_NC ISPASMV,<ZF10LP,ZF10_PARM,EZVU_RC,ZF10,ZF10LV>
        DMPC_NC ISPASMV,<ZF11LP,ZF11_PARM,EZVU_RC,ZF11,ZF11LV>
        DMPC_NC ISPASMV,<ZF12LP,ZF12_PARM,EZVU_RC,ZF12,ZF12LV>
        test    Active_Keys, F1_OK      ;Is F1 an invalid key?
        jne     chk_f2_ok               ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF01_A  ;Set key inactive
        call    delete_zfkey
chk_f2_ok:
        test    Active_Keys, F2_OK      ;Is this an invalid key?
        jne     chk_f3_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF02_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f3_ok:
        test    Active_Keys, F3_OK    ;Is this an invalid key?
        jne     chk_f4_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF03_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f4_ok:
        test    Active_Keys, F4_OK    ;Is this an invalid key?
        jne     chk_f5_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF04_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f5_ok:
        test    Active_Keys, F5_OK    ;Is this an invalid key?
        jne     chk_f6_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF05_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f6_ok:
        test    Active_Keys, F6_OK    ;Is this an invalid key?
        jne     chk_f7_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF06_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f7_ok:
        test    Active_Keys, F7_OK    ;Is this an invalid key?
        jne     chk_f8_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF07_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f8_ok:
        test    Active_Keys, F8_OK    ;Is this an invalid key?
        jne     chk_f9_ok             ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF08_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f9_ok:
        test    Active_Keys, F9_OK    ;Is this an invalid key?
        jne     chk_f10_ok            ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF09_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f10_ok:
        test    Active_Keys, F10_OK   ;Is this an invalid key?
        jne     chk_f11_ok            ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF10_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f11_ok:
        test    Active_Keys, F11_OK   ;Is this an invalid key?
        jne     chk_f12_ok            ;If not, go check next key
        mov     ZFKEY_TO_DELETE,ZF11_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it
chk_f12_ok:
        test    Active_Keys, F12_OK   ;Is this an invalid key?
        jne     chk_other_keys        ;If not, go check other keys
        mov     ZFKEY_TO_DELETE,ZF12_A ;Set key to be deactivated
        call    delete_zfkey          ;And VDELETE it

;*******************************************************************
;After this point, we are looking at keys that need to have their
;       scan codes placed in the ZENT variable to be recognized.
;*******************************************************************
chk_other_keys:
        FILL_CHAR KEYLINE,' ',Keylinelv ;Clear out text for non-F keys
        FILL_CHAR ZENT1,0,LNGTH9V       ;Clear out list of valid Enter keys

        mov     di, offset KEYLINE      ;Reset output pointer
        push    ds
        pop     es
        xor     bx,bx                   ; Used for offset into ZENT array

        test    Active_Keys, ENTER_OK   ;Do we want this key?
        je      chk_next_1              ;If not, check next key
        mov     ax, word ptr ZENT1n     ;If so, get its scan code
        mov     word ptr ZENT1[bx], ax  ;And put it in the EZVU array
        add     bx, 2
        mov     si, offset Enter_Text   ;Move text of key in
        mov     cx, ENTER_TLEN
        rep     movsb

chk_next_1:
        test    Active_Keys, ESC_OK     ;Do we want this key?
        je      chk_next_2              ;If not, check next key
        mov     ax, word ptr ZENT1E     ;If so, get its scan code
        mov     word ptr ZENT1[bx], ax  ;And put it in the EZVU array
        add     bx, 2

;       mov     si, offset Escape_Text  ;Move text of Esc key in
;       mov     cx, ESCAPE_TLEN
;       rep     movsb

chk_next_2:
        test    Active_Keys, PGUP_OK    ;Do we want this key?
        je      chk_next_3              ;If not, check next key
        mov     ax, word ptr ZENT1PUP   ;If so, get its scan code
        mov     word ptr ZENT1[bx], ax  ;And put it in the EZVU array
        add     bx, 2
        mov     si, offset PgUp_Text    ;Move text of PgUp key in
        mov     cx, PGUP_TLEN
        rep     movsb

chk_next_3:
        test    Active_Keys, PGDN_OK    ;Do we want this key?
        je      chk_keys_ok_exit        ;If not, check next key
        mov     ax, word ptr ZENT1PDN   ;If so, get its scan code
        mov     word ptr ZENT1[bx], ax  ;And put it in the EZVU array
        add     bx, 2
        mov     si, offset PgDn_Text    ;Move text of PgDn key in
        mov     cx, PGDN_TLEN
        rep     movsb

chk_keys_ok_exit:
        POPREGS
        ret
Set_Active_Keys endp

;*******************************************************************
; delete_zfkey
;
; Input:
;       ZFKEY_TO_DELETE - Set to ASCII Fkey to delete (e.g. '01', '12')
;
```

```
; Output:
;       Appropriate F key deleted from EZVU pools
;*********************************************************************
delete_zfkey    proc    near
        DMPC_NC ISPASM,<ZFKEY_DELETEL, ZFKEY_DELETE, EZVU_RC>
        ret
delete_zfkey    endp

;******************************************************************
;*                                                                *
;*  Procedure Name: Hexb2asc                                      *
;*                                                                *
;*  Description  : Converts the value contained in AL to a two    *
;*                 character Hex/ASCII string pointed at by DI.   *
;*                                                                *
;*  Input : Register DI points at a 2 char string buffer to be    *
;*          used as the target buffer for the conversion.         *
;*          Register AL contains value to be converted.           *
;*                                                                *
;*  Output : The target buffer pointed at by DI will contain      *
;*           a two  char Hex-ASCII string that is the             *
;*           representation of the value passed in AX.            *
;*                                                                *
;******************************************************************

PUBLIC Hexb2asc
Hexb2asc proc near
         push    ax                            ;save ax


         sub     ah, ah                        ;zero out high byte
         push    di                            ;save di
         lea     di,hexb2asc_buff              ;point di at target buffer
                                               ;for Hex2asc
         call    Hex2asc                       ;convert to a four byte string
         pop     di                            ;restore di
         mov     ax,word ptr hexb2asc_buff+2   ;put last 2 of 4 converted chars
                                               ;chars in ax
         mov     word ptr [di],ax              ;store 2 chars in caller's buffer

         pop     ax                            ;restore ax

         ret

Hexb2asc ENDP

CSEG    ENDS


end
```

APIDISP.DSG

```
;       API Sample Program - (C) Copyright IBM Corp. 1986, 1987
;       SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.
;

DGROUP  GROUP   DATA,STACK

STACK   SEGMENT BYTE STACK 'STACK'
        DB      256 DUP('STACK   ')      ; 2K STACK AREA
STKTOP  DW      1
STACK   ENDS

DATA    SEGMENT PARA PUBLIC 'DATA'
        ASSUME  DS:DGROUP


        CR_LF  EQU WORD PTR 0A0DH      ; ASCII Code for Carriage Return/Line Feed
        CR     EQU BYTE PTR 13D        ; ASCII Code for Carriage Return
        ESC    EQU BYTE PTR 27D        ; ASCII Code for Escape Code
        F1     EQU BYTE PTR 59D        ; Scan Code for F1  key
        F2     EQU BYTE PTR 60D        ; Scan Code for F2  key
        F3     EQU BYTE PTR 61D        ; Scan Code for F3  key
        F4     EQU BYTE PTR 62D        ; Scan Code for F4  key
        F5     EQU BYTE PTR 63D        ; Scan Code for F5  key
        F6     EQU BYTE PTR 64D        ; Scan Code for F6  key
        F7     EQU BYTE PTR 65D        ; Scan Code for F7  key
        F8     EQU BYTE PTR 66D        ; Scan Code for F8  key
        F9     EQU BYTE PTR 67D        ; Scan Code for F9  key
        F10    EQU BYTE PTR 68D        ; Scan Code for F10 key
        F11    EQU BYTE PTR 84D        ; Scan Code for F11 key
        F12    EQU BYTE PTR 85D        ; Scan Code for F12 key
        PAGEUP EQU BYTE PTR 73d        ; Scan Code for Page up   key
        PAGEDN EQU BYTE PTR 81d        ; Scan Code for Page down key


ZENTF9n    DB F9            ; F9  key - scan  code
ZENTF9nb   DB 0             ; F9  key - ASCII code


ZENTF11n   DB F11           ; F11 key - scan  code
ZENTF11b   DB 0             ; F11 key - ASCII code

PARM10D   DB    'CONTROL CURSOR '
ZFLD      DB    '        '
ZCRS      DB    '  '
LNGTH10PD DW    LNGTH10PD - PARM10D
```

```
PARM8D    DB    'DISPLAY'
LNGTH8PD  DW    LNGTH8PD - PARM8D

;*********************************************************************
; Key definitions for some panels
;*********************************************************************

DCJVCX00_KEYS   EQU F3_OK+F9_OK+ESC_OK
DCJVBP03_KEYS   EQU F3_OK+F6_OK+F9_OK+F11_OK+F12_OK+PGUP_OK+PGDN_OK+ESC_OK
DCJVBP04_KEYS   EQU F3_OK+F6_OK+F9_OK+ESC_OK+F12_OK

;*********************************************************************
; Panel names
;*********************************************************************
DCJVBP03  DB    'DISPLAY DCJVBP03'            ;EZVU command
DCJVBP03L DW    DCJVBP03L - DCJVBP03

DCJVBP04  DB    'DISPLAY DCJVBP04'            ;EZVU command
DCJVBP04L DW    DCJVBP04L - DCJVBP04

DCJVCX00  DB    'DISPLAY DCJVCX00'            ;EZVU command
DCJVCX00L DW    DCJVCX00L - DCJVCX00

DCJVBPPD  DB    'PANDEL'                      ;Delete panels
DCJVBPPDL DW    DCJVBPPDL - DCJVBPPD

PJTITLE_LENGTH EQU 70          ;Length of title field
;Title field on panel
PJTITLEC_PARM   DB      'PJTITLEC C'
PJTITLELP       DW      PJTITLELP-PJTITLEC_PARM
PJTITLEC        DB      PJTITLE_LENGTH DUP(' ')
PJTITLELV       DW      PJTITLELV-PJTITLEC


;Display type:  Note that 0=unformatted 1=ASCII, 2=EBCDIC
;       If this is changed, must change translation on the panel
PJDSPTYP_PARM   DB      'PJDSPTYP C'
PJDSPTYPLP      DW      PJDSPTYPLP-PJDSPTYP_PARM
PJDSPTYP        DB      '1'
PJDSPTYPLV      DW      PJDSPTYPLV-PJDSPTYP




;********************************************************************
;* Dump (Unformatted) panel variables
;********************************************************************

PJTITLE_A       DB      '     VENDOR API SPCF DISPLAY FILE    Display Type: Dump (ASCII)       '
PJTITLE_E       DB      '     VENDOR API SPCF DISPLAY FILE    Display Type: Dump (EBCDIC)      '


;File name
PJFILENC_PARM   DB      'PJFILENC C'
PJFILENLP       DW      PJFILENLP-PJFILENC_PARM
PJFILENC        DB      'NMVTFILE.BIN  '
PJFILENLV       DW      PJFILENLV-PJFILENC-2

PUBLIC PJFILENC
```

```
;Other type of character conversion (goes next to F key)
PJOTYPEC_PARM   DB      'PJOTYPEC C'
PJOTYPELP       DW      PJOTYPELP-PJOTYPEC_PARM
PJOTYPEC        DB      'EBCDIC'
PJOTYPELV       DW      PJOTYPELV-PJOTYPEC

PJOTYPE_A       DB      'ASCII '
PJOTYPE_E       DB      'EBCDIC'

;length of dump field
PJDMPLNC_PARM   DB      'PJDMPLNC C'
PJDMPLNLP       DW      PJDMPLNLP-PJDMPLNC_PARM
PJDMPLNC        DB      '0000'
PJDMPLNLV       DW      PJDMPLNLV-PJDMPLNC

;Starting offset input field
PJOFFSEC_PARM   DB      'PJOFFSEC C'
PJOFFSELP       DW      PJOFFSELP-PJOFFSEC_PARM
PJOFFSECX       DB     '0000' ;*** Look Below for real value
PJOFFSELV       DW      PJOFFSELV-PJOFFSECX

NLINES          EQU     16       ;# of lines on display
COLUMNS         EQU     16       ;Bytes displayed per line
PAGESIZE        EQU     NLINES * COLUMNS       ;Bytes displayed per page

;Rest of offset input field
PJOFFSTC_PARM   DB      'PJOFFSTC C'
PJOFFSTLP       DW      PJOFFSTLP-PJOFFSTC_PARM
PJOFFSEC        DB      '0000'          ;So we treat offsets the same
PJOFFSTC        DB      NLINES-1 DUP('0000')
PJOFFST_BYTES   DW      4               ;bytes in an offset field thingy
PJOFFST_SIZE    DW      NLINES-1        ;Number of elements
PJOFFST_VI      DW      0               ;Vertical index
PJOFFST_HI      DW      0               ;Horizontal index

;Hexadecimal dump area
PJDMPHXC_PARM   DB      'PJDMPHXC C'
PJDMPHXLP       DW      PJDMPHXLP-PJDMPHXC_PARM
PJDMPHXC        DB      NLINES DUP('0000 0000 0000 0000  0000 0000 0000 0000')
PJDMPHX_BYTES   DW      40              ;bytes in an hex dump line
PJDMPHX_SIZE    DW      NLINES          ;Number of elements
PJDMPHX_VI      DW      0               ;Vertical index
PJDMPHX_HI      DW      0               ;Horizontal index

;Character dump area
PJDMPCHC_PARM   DB      'PJDMPCHC C'
PJDMPCHLP       DW      PJDMPCHLP-PJDMPCHC_PARM
PJDMPCHC        DB      NLINES DUP('........ ........')
PJDMPCH_BYTES   DW      17              ;bytes in an character dump line
PJDMPCH_SIZE    DW      NLINES          ;Number of elements
PJDMPCH_VI      DW      0               ;Vertical index
PJDMPCH_HI      DW      0               ;Horizontal index
```

```
;***********************************************************************
;* Dump (Formatted) panel variables
;***********************************************************************

PJTITLE_RUN     DB      '      VENDOR API SPCF DISPLAY RUN COMMAND    Display Type:  Formatted '
PJTITLE_LPD     DB      '  VENDOR API SPCF DISPLAY LINK PD COMMAND    Display Type:  Formatted '
PJTITLE_LT      DB      'VENDOR API SPCF DISPLAY LINK TEST COMMAND    Display Type:  Formatted '
PJTITLE_LD      DB      'VENDOR API SPCF DISPLAY LINK DATA COMMAND    Display Type:  Formatted '
PJTITLE_NULL    DB      '         VENDOR API SPCF FORMATTED DISPLAY - No Display Active        '


;ARB ID
P04ARBID_PARM   DB      'P04ARBID C'
P04ARBIDLP      DW      P04ARBIDLP-P04ARBID_PARM
P04ARBID        DB      'ARB6'
P04ARBIDLV      DW      P04ARBIDLV-P04ARBID

;Request code
P04RQCOD_PARM   DB      'P04RQCOD C'
P04RQCODLP      DW      P04RQCODLP-P04RQCOD_PARM
P04RQCOD        DB      '0000'
P04RQCODLV      DW      P04RQCODLV-P04RQCOD

;ARB Length
P04ARBLN_PARM   DB      'P04ARBLN I'
P04ARBLNLP      DW      P04ARBLNLP-P04ARBLN_PARM
P04ARBLN        DB      36
P04ARBLNLV      DW      P04ARBLNLV-P04ARBLN

;Parse ID
P04PRSID_PARM   DB      'P04PRSID C'
P04PRSIDLP      DW      P04PRSIDLP-P04PRSID_PARM
P04PRSID        DB      '61'
P04PRSIDLV      DW      P04PRSIDLV-P04PRSID

;Receive correlator
P04RCVCR_PARM   DB      'P04RCVCR C'
P04RCVCRLP      DW      P04RCVCRLP-P04RCVCR_PARM
P04RCVCR        DB      '0123456789abcdef0123'
P04RCVCRLV      DW      P04RCVCRLV-P04RCVCR

;Return code
P04RETCD_PARM   DB      'P04RETCD I'
P04RETCDLP      DW      P04RETCDLP-P04RETCD_PARM
P04RETCD        DW      -1
P04RETCDLV      DW      P04RETCDLV-P04RETCD

;Error Class
P04ERCLS_PARM   DB      'P04ERCLS I'
P04ERCLSLP      DW      P04ERCLSLP-P04ERCLS_PARM
P04ERCLS        DW      -1
P04ERCLSLV      DW      P04ERCLSLV-P04ERCLS

;Error Type
P04ERTYP_PARM   DB      'P04ERTYP I'
P04ERTYPLP      DW      P04ERTYPLP-P04ERTYP_PARM
P04ERTYP        DW      -1
P04ERTYPLV      DW      P04ERTYPLV-P04ERTYP
```

```
;Parse Sense Data
P04PRSNS_PARM   DB      'P04PRSNS C'
P04PRSNSLP      DW      P04PRSNSLP-P04PRSNS_PARM
P04PRSNS        DB      '01234567'      ;Parse sense data
P04PRSNSLV      DW      P04PRSNSLV-P04PRSNS

;Command length and Number of resources field
P04CMDLN_PARM   DB      'P04CMDLN I'
P04CMDLNLP      DW      P04CMDLNLP-P04CMDLN_PARM
P04CMDLN        DB      0
P04CMDLNLV      DW      P04CMDLNLV-P04CMDLN

;Command length Text (Command Length: or Number of Resources:)
P04CMDTX_PARM   DB      'P04CMDTX C'
P04CMDTXLP      DW      P04CMDTXLP-P04CMDTX_PARM
P04CMDTX        DB      'Command Length. . .:'
P04CMDTXLV      DW      P04CMDTXLV-P04CMDTX

P04CMDTX_Command        DB   'Command Length. . .:'
P04CMDTX_Resources      DB   'Number of Resources:'
P04CMDTX_Null           DB   'Length of Data. . .:'

;Command Text for command
P04CMDLI_PARM   DB      'P04CMDLI C'
P04CMDLILP      DW      P04CMDLILP-P04CMDLI_PARM
P04CMDLI        DB      'Command:'
P04CMDLILV      DW      P04CMDLILV-P04CMDLI

;Resources list
P04RESRC_PARM   DB      'P04RESRC C'
P04RESRCLP      DW      P04RESRCLP-P04RESRC_PARM
P04RESRC        DB      'Resources:'
P04RESRCLV      DW      P04RESRCLV-P04RESRC

;Test Count text
P04TSCNX_PARM   DB      'P04TSCNX C'
P04TSCNXLP      DW      P04TSCNXLP-P04TSCNX_PARM
P04TSCNX        DB      'Test Count: '
P04TSCNXLV      DW      P04TSCNXLV-P04TSCNX

;Test Count data
P04TSCNT_PARM   DB      'P04TSCNT I'
P04TSCNTLP      DW      P04TSCNTLP-P04TSCNT_PARM
P04TSCNT        DW      0
P04TSCNTLV      DW      P04TSCNTLV-P04TSCNT

;Test Type text
P04TSTYX_PARM   DB      'P04TSTYX C'
P04TSTYXLP      DW      P04TSTYXLP-P04TSTYX_PARM
P04TSTYX        DB      'Test Type: '
P04TSTYXLV      DW      P04TSTYXLV-P04TSTYX

;Test Type data
P04TSTYP_PARM   DB      'P04TSTYP I'
P04TSTYPLP      DW      P04TSTYPLP-P04TSTYP_PARM
P04TSTYP        DB      0
P04TSTYPLV      DW      P04TSTYPLV-P04TSTYP
```

```
P04_NLINES      EQU     5
P04_COLUMNS     EQU     64

P04_SENSE_LEN   EQU     4       ;Number of bytes in sense length

;The big data field
P04RDATA_PARM   DB      'P04RDATA C'
P04RDATALP      DW      P04RDATALP-P04RDATA_PARM
P04RDATA        DB      P04_COLUMNS dup (' ')
P04RDATA2       DB      (P04_NLINES-1)*P04_COLUMNS dup (' ')
P04RDATALV      DW      P04RDATALV-P04RDATA
P04RDATA_BYTES  DW      P04_COLUMNS     ;Number of characters in a line
P04RDATA_SIZE   DW      P04_NLINES      ;Number of lines
P04RDATA_VI     DW      0               ;Vertical index
P04RDATA_HI     DW      0               ;Horizontal index

P04_COLLINE     DB      '....+....1....+....2....+....3....+....4....+....5....+....6....'

;*********************************************************************
;The following are the commands used to VDELETE some of the above in
;       order to customize the panel for the various parse IDS
;*********************************************************************
P04TSCNX_DELETE DB      'VDELETE P04TSCNX A'
P04TSCNT_DELETE DB      'VDELETE P04TSCNT A'
P04TSTYX_DELETE DB      'VDELETE P04TSTYX A'
P04TSTYP_DELETE DB      'VDELETE P04TSTYP A'
P04CMDLI_DELETE DB      'VDELETE P04CMDLI A'
P04RESRC_DELETE DB      'VDELETE P04RESRC A'
P04_DELETE_LEN  DW      P04_DELETE_LEN - P04RESRC_DELETE


;*********************************************************************
;* Display tables for unformatted dump
;*********************************************************************
ASCII_DISPLAY_TABLE EQU BYTE PTR $
 ;          0123456789abcdef
        DB '................'  ;00
        DB '................'  ;10
        DB ' !"#$%&',27h,'()*+',2ch,'-./'  ;20
        DB '0123456789:;<=>?'  ;30

 ;          0123456789abcdef
        DB '@ABCDEFGHIJKLMNO'  ;40
        DB 'PQRSTUVWXYZ[\]._'  ;50
        DB '.abcdefghijklmno'  ;60
        DB 'pqrstuvwxyz{|}..'  ;70

 ;          0123456789abcdef
        DB '................'  ;80
        DB '................'  ;90
        DB '................'  ;A0
        DB '................'  ;B0

 ;          0123456789abcdef
        DB '................'  ;C0
        DB '................'  ;D0
        DB '................'  ;E0
        DB '................'  ;F0
```

```
EBCDIC_DISPLAY_TABLE EQU BYTE PTR $
 ;          0123456789abcdef
        DB '................'  ;00
        DB '................'  ;10
        DB '................'  ;20
        DB '................'  ;30

 ;          0123456789abcdef
        DB ' .........¢.<(+|'  ;40
        DB '&.........!$*);.'  ;50
        DB '-/.........,%_>?'  ;60
        DB '..........:#@',2ch,'=',22h ;70

 ;          0123456789abcdef
        DB '.abcdefghi.{....'  ;80
        DB '.jklmnopqr.}....'  ;90
        DB '..stuvwxyz.....„'  ;A0
        DB '..............._'  ;B0

 ;          0123456789abcdef
        DB '.ABCDEFGHI......'  ;C0
        DB '.JKLMNOPQR......'  ;D0
        DB '.\STUVWXYZ......'  ;E0
        DB '0123456789......'  ;F0

;***********************************************************************
; Work variables for unformatted dump display
;***********************************************************************

pj_bufsize      equu    4096d
;Buffer is defined below, overlaying of the parse arb structure

pj_datsize      dw      0               ;Amount of data in buffer
pj_nlines       dw      16              ;Number of lines
pj_remainder    dw      0
pj_offset       dw      0               ;Offset into buffer
pj_offset_save  dw      0               ;Offset saved
pj_translate_fg dw      ASCII_FG        ;Translation (ASCII or EBCDIC)

PUBLIC pj_translate_fg

pj_offset_ptr   dw      0               ;Present offset array location
pj_dumphx_ptr   dw      0               ;Present hex dump array location
pj_dumpch_ptr   dw      0               ;Present character dump location

pj_translate_ptr dw     ASCII_DISPLAY_TABLE     ;pointer to xlate table

PUBLIC ASCII_FG
PUBLIC EBCDIC_FG

ASCII_FG        EQU 0
EBCDIC_FG       EQU 1


EMPTY   EQU     60909           ;Used to indicate past end of data
                                ;       = 'ōō'
EMPTYC  EQU     237             ;Same for character display
```

```
SPACEC  EQU     ' '             ;For formatted displays, filler char.

;***********************************************************************
; Parse ARB Structure definition
;***********************************************************************

parse_arb       struc
pj_arbid        db      'ARB6'
pj_reqcode      dw      0h
pj_arblen       db      36
pj_parseid      db      0
pj_reserved     db      0
pj_retcode      dw      0
pj_errclass     dw      0
pj_errtype      dw      0
pj_parsenmvt    dd      0
pj_numnames     db      0
pj_names        dd      0
pj_testcount    dw      0
pj_testtype     db      0
pj_sensedata    db      P04_SENSE_LEN dup(0)
pj_commandlen   db      0
pj_command      dd      0
pj_recvcorr     db      10 dup(0)
pj_parsedata    db      pj_bufsize dup(?)
parse_arb       ends

pj_arb          equ     Arbbuff            ;Allocate storage

P04_request_code        equ     0          ;Parse request code

;***********************************************************************
; Define dispatch tables for the formatted display
;***********************************************************************
p04_vdef_dispatch       equ $           ;Dispatch table for VDEFINES
                dw      0
                dw      define_run      ;Vdefines for run command display
                dw      define_link_pd  ;Vdefines for link pd display
                dw      define_link_data;Vdefines for link data display
                dw      define_link_test;Vdefines for link test display

p04_vdel_dispatch       equ $           ;Dispatch table for VDELETES
                dw      0
                dw      delete_run      ;Vdeletes for run command display
                dw      delete_link_pd  ;Vdeletes for link pd display
                dw      delete_link_data;Vdeletes for link data display
                dw      delete_link_test;Vdeletes for link test display

p04_form_dispatch       equ $           ;Dispatch table for format procedures
                dw      0
                dw      format_run      ;Format data for run command display
                dw      format_link     ;Format data for link pd display
                dw      format_link     ;Format data for link data display
                dw      format_link     ;Format data for link test display

p04_titles              equ $           ;Pointers to titles for ARBs
                dw      PJTITLE_NULL    ;Put up when error reading file
                dw      PJTITLE_RUN     ;Title line for run command display
                dw      PJTITLE_LPD     ;Title line for link pd display
                dw      PJTITLE_LD      ;Title line for link data display
                dw      PJTITLE_LT      ;Title line for link test display

P04_MIN_PID     db      61h             ;Minimum parse ID recognized
P04_NUM_PID     db      4h              ;Maximum number of parse IDs recognized
P04_jump_offset dw      0               ;Used to save calculated offset

;*********************************************************************
;Error message numbers
;
PJ_NON_HEX      EQU     200D            ;Non-hex in offset field
PJ_BEGINNING    EQU     201D            ;Now at beginning of dump
PJ_ENDING       EQU     202D            ;Now at end of dump
PJ_BAD_OFFSET   EQU     203D            ;Offset past end of buffer
PJ_FORMAT_NA    EQU     204D            ;Formatted dump not available
P04_BAD_REQCODE EQU     205d            ;Bad request code
P04_BAD_ARBID   EQU     206d            ;Bad ARB ID  code
P04_BAD_ARBLEN  EQU     207d            ;Bad length
P04_BAD_PARSEID EQU     208d            ;Bad parse ID

;*********************************************************************
;SCAN CODES
        ENTER EQU BYTE PTR 1ch        ; Scan Code for Enter


;*********************************************************************
;MISCELLANEOUS

pj_fieldnamel   equ     8               ;Lengths of field names

pj_filename_old DB      'NMVTFILE.BIN  ';Place to save old file name

pj_cross        db      0               ;Set if going from formatted to
                                        ;       unformatted display or
                                        ;       vice versa

DATA    ENDS
```

APIDISP.ASM

```
; (CTRL-OH) IBM PC PRINTER CONDENSED MODE
         PAGE  ,132
         TITLE API Sample Program - (C) Copyright IBM Corp. 1986,1987
;        SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
;
;   You are hereby licensed to use, reproduce, and distribute
;   these sample programs as your needs require.  IBM does not
;   warrant the suitability or integrity of these sample programs
;   and accepts no responsibility for their use for your
;   applications.  If you choose to copy and redistribute
;   significant portions of these sample programs, you should
;   preface such copies with this copyright notice.
;
         .SALL                           ;Suppress macro expansion
```

```
        INCLUDE  APIMAIN.DEF            ;Constant definitions
        INCLUDE  APIDISP.DSG            ;Data Segment and references
        INCLUDE  APIUTIL.EXR            ;External References from APIUTIL
        INCLUDE  APIMAIN.EXR            ;External References from APIMAIN
         IF1
           INCLUDE  APIMAIN.MAC          ;Macros
         ELSE
          %OUT Starting second pass ...
        ENDIF


PAGE

PGROUP  GROUP   CSEG

        PUBLIC  SPCF_DISPLAY_INIT       ;Routine to define things to EZ-VU
        PUBLIC  SPCF_DISPLAY_PAN        ;Main display procedure

CSEG    SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:PGROUP,DS:DGROUP,ES:DGROUP,SS:NOTHING

        EXTRN ISPASM:FAR                ;EZ-VU II Display functions
        EXTRN ISPASMV:FAR               ;EZ-VU II Variable definitions
        EXTRN ISPASMVA:FAR              ;EZ-VU II Variable definitions


;*************************************************************************
; spcf_display_init - VDEFINE display variables to EZ-VU
;
;*************************************************************************
spcf_display_init       proc    near
        pushregs

;*************************************************************************
;Define variables needed for selection
;*************************************************************************

        DMPC_NS ISPASMV,<PJDSPTYPLP,PJDSPTYP_PARM,EZVU_RC,PJDSPTYP,PJDSPTYPLV>

;*************************************************************************
;Define variables needed for unformatted dump panel panel
;*************************************************************************

        DMPC_NS ISPASMV,<PJTITLELP,PJTITLEC_PARM,EZVU_RC,PJTITLEC,PJTITLELV>
        DMPC_NS ISPASMV,<PJFILENLP,PJFILENC_PARM,EZVU_RC,PJFILENC,PJFILENLV>
        DMPC_NS ISPASMV,<PJDMPLNLP,PJDMPLNC_PARM,EZVU_RC,PJDMPLNC,PJDMPLNLV>
        DMPC_NS ISPASMV,<PJOFFSELP,PJOFFSEC_PARM,EZVU_RC,PJOFFSEC,PJOFFSELV>

        DMPC_NS
ISPASMVA,<PJOFFSTLP,PJOFFSTC_PARM,EZVU_RC,PJOFFSTC,PJOFFST_BYTES,PJOFFST_SIZE,PJOFFST_VI,PJOFFST_HI>
        DMPC_NS
ISPASMVA,<PJDMPHXLP,PJDMPHXC_PARM,EZVU_RC,PJDMPHXC,PJDMPHX_BYTES,PJDMPHX_SIZE,PJDMPHX_VI,PJDMPHX_HI>
        DMPC_NS
ISPASMVA,<PJDMPCHLP,PJDMPCHC_PARM,EZVU_RC,PJDMPCHC,PJDMPCH_BYTES,PJDMPCH_SIZE,PJDMPCH_VI,PJDMPCH_HI>
        DMPC_NS ISPASMV,<PJOTYPELP,PJOTYPEC_PARM,EZVU_RC,PJOTYPEC,PJOTYPELV>
```

```
;*********************************************************************
;Now those for the formatted dumps
;*********************************************************************
        DMPC_NS ISPASMV,<P04ARBIDLP,P04ARBID_PARM,EZVU_RC,P04ARBID,P04ARBIDLV>
        DMPC_NS ISPASMV,<P04RQCODLP,P04RQCOD_PARM,EZVU_RC,P04RQCOD,P04RQCODLV>
        DMPC_NS ISPASMV,<P04ARBLNLP,P04ARBLN_PARM,EZVU_RC,P04ARBLN,P04ARBLNLV>
        DMPC_NS ISPASMV,<P04PRSIDLP,P04PRSID_PARM,EZVU_RC,P04PRSID,P04PRSIDLV>
        DMPC_NS ISPASMV,<P04RCVCRLP,P04RCVCR_PARM,EZVU_RC,P04RCVCR,P04RCVCRLV>
        DMPC_NS ISPASMV,<P04RETCDLP,P04RETCD_PARM,EZVU_RC,P04RETCD,P04RETCDLV>
        DMPC_NS ISPASMV,<P04ERCLSLP,P04ERCLS_PARM,EZVU_RC,P04ERCLS,P04ERCLSLV>
        DMPC_NS ISPASMV,<P04ERTYPLP,P04ERTYP_PARM,EZVU_RC,P04ERTYP,P04ERTYPLV>
        DMPC_NS ISPASMV,<P04PRSNSLP,P04PRSNS_PARM,EZVU_RC,P04PRSNS,P04PRSNSLV>
        DMPC_NS ISPASMV,<P04CMDLNLP,P04CMDLN_PARM,EZVU_RC,P04CMDLN,P04CMDLNLV>
        DMPC_NS ISPASMV,<P04CMDTXLP,P04CMDTX_PARM,EZVU_RC,P04CMDTX,P04CMDTXLV>
        DMPC_NS
ISPASMVA,<P04RDATALP,P04RDATA_PARM,EZVU_RC,P04RDATA,P04RDATA_BYTES,P04RDATA_SIZE,P04RDATA_VI,P04RDATA_HI>


;*********************************************************************
; Key display
;*********************************************************************
        DMPC_NS ISPASMV,<Keylinelp,Keyline_Parm,EZVU_RC,Keyline,Keylinelv>

        popregs
        ret
spcf_display_init endp

;*********************************************************************
; spcf_display_pan - Pop up requesting filename to display
;
;*********************************************************************
spcf_display_pan        proc    near
        pushregs
;Use the ARB file name as the default, if next line uncommented
;       MOVE_STRING Arbfile, PJFILENC, PJFILENLV
display_fn_req:
        mov     Active_Keys, DCJVCX00_KEYS      ;Set active keys
        call    Set_Active_Keys
        DMPC    ISPASM,<DCJVCX00L,DCJVCX00,EZVU_RC>

display_fn_req_loop:
         cmp    Zrspl,F3                        ;Was F3 the exit key?
         jne    not_f3_dsp
         jmp    exit_display_fn_req             ;Yes, Exit this rtn

not_f3_dsp:
         cmp    Zrspl,F9                        ;Was it display request
         jne    display_fn_req_refresh          ;If not, refresh panel
         jmp    do_spcf_display                 ;If so, display it

do_spcf_display:
         mov    al, PJDSPTYP                    ;Get field value
         sub    ah, ah                          ;Clear high byte
         sub    al, '0'                         ;Turn it into a number
         je     spcf_formatted                  ;If 0,  was formatted
                                                ; 1 = ASCII
                                                ; 2 = EBCDIC
         dec    ax                              ;ax now has either 0 or 1
         mov    pj_translate_fg, ax             ;And store it
         call    spcf_display_unformatted       ;Else do unformatted display
         jmp     exit_display_fn_req            ;Then exit

spcf_formatted:
         call    spcf_display_formatted         ;Else do formatted display
         jmp     exit_display_fn_req            ;Then exit

display_fn_req_refresh:
        DMPC    ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>   ;Else redisplay panel
        jmp     display_fn_req_loop

exit_display_fn_req:
        popregs
        ret
spcf_display_pan  endp

;*********************************************************************
; spcf_display_unformatted - Unformatted file display
;
; This is the panel handler for the unformatted display.
;
; Input:
;     PJFILENC        - file to be displayed
;     pj_translate_fg - Translation type (ASCII/EBCDIC) (0/1)
;
; Output:
;     PJFILENC        - may be modified by user
;
;*********************************************************************
PUBLIC spcf_display_unformatted
spcf_display_unformatted proc near
         pushregs
do_panel_read:
         call    read_in_file
do_panel_format:
         mov    pj_offset, 0                    ;Set initial offset to 0

tot_display_do_panel:
         call   format_data
         call   set_unformatted_keys
         DMPC   ISPASM,<DCJVBP03L,DCJVBP03,EZVU_RC>

display_do_panel:
         cmp     Zrspl,F3                       ;Was F3 the exit key?
         jne     not_f3_3m
         call    restore_filenm                 ;Restore old file name
         mov     pj_cross, 0                    ;Clear out crossing flag
         jmp     exit_do_panel                  ;Yes, Exit this rtn

not_f3_3m:
         cmp     Zrspl,F6                       ;Was it DOS request?
         jne     not_f6_3m
         call    restore_filenm                 ;Restore old file name
         call    execpgm                        ;Invoke second command
                                                ;      processor

         call    read_in_file                   ;On exit, re-read file
                                                ; and redisplay
        cmp     read_nmvt_stat, 0               ;Check for success
        je      file_readok_dosexit_3m          ;Was successful, branch around

        mov     pj_offset,0                     ;Set initial offset to 0
        jmp     tot_display_do_panel            ;Then redisplay

file_readok_dosexit_3m:
        call    setup_offset                    ;Go check offset
        jmp     tot_display_do_panel            ;Then redisplay


not_f6_3m:
        cmp     Zrspl,F12                       ;Was it format display?
        jne     not_f12_3m                      ;If not, check next
        call    restore_filenm                  ;Restore old file name
        cmp     pj_cross, 0                     ;Did we come from formatted?
        jne     unformatted_cross               ;If non-zero, we did
        mov     pj_cross, 1                     ;Otherwise, we didn't, so set
        call    spcf_display_formatted          ;Else format display

        cmp     Zrspl,F3                        ;Was F3 the exit key from
                                                ;the formatted display panel?
        jne     back_to_unformatted             ;No, returned to this panel.
        jmp     exit_do_panel                   ;Yes, wanted to quit so exit

back_to_unformatted:
        jmp     do_panel_read                   ;redisplay unformatted panel

unformatted_cross:                              ;We came from formatted, so
       mov     pj_cross, 0                      ;       clear the flag and
       jmp     exit_do_panel                    ;       exit this panel

not_f12_3m:
        cmp     Zrspl,F11                       ;Was it change translation?
        jne     not_f11_3m                      ;If not, check next
        call    restore_filenm                  ;Restore old file name
        xor     pj_translate_fg, ebcdic_fg      ;Toggle flag
        jmp     refresh_do_panel                ;And loop

not_f11_3m:
        cmp     Zrspl,F8                        ;Was F8 the exit key?
        jne     not_f8_3m                       ;No
        call    restore_filenm                  ;Restore old file name
        jmp     exit_do_panel                   ;Yes, do whatever
                                                ;SPCF main panel.

not_f8_3m:
       cmp     Zrspl,F9                         ;Was it F9?
       jne     not_F9_3m                        ;If not, go check next key

       call    read_in_file
       cmp     read_nmvt_stat, 0                ;Check for success
       je      file_readok_3m

       mov     pj_offset,0                      ;Set initial offset to 0
       jmp     refresh_do_panel                 ;Then redisplay

file_readok_3m:
       call    setup_offset                     ;Go check offset
```

```
        jmp     refresh_do_panel                ;Then redisplay

not_F9_3m:
        cmp     Zrspl, pageup                   ;Was it page up?
        jne     not_pup_3m                      ;No
        call    restore_filenm                  ;Restore old file name
        call    do_pgup                         ;Do a page up
        jmp     refresh_do_panel                ;And redisplay

not_pup_3m:
        cmp     Zrspl, pagedn                   ;Was it page up?
        jne     redisplay_do_panel              ;No, redisplay panel
        call    restore_filenm                  ;Restore old file name
        call    do_pgdn                         ;Do a page up
        jmp     refresh_do_panel                ;And redisplay

refresh_do_panel:
        call    format_data                     ;Do new data

redisplay_do_panel:
        call    set_unformatted_keys
        DMPC ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>   ;Else redisplay
        jmp     display_do_panel                ;And redisplay panel

PAGE

exit_do_panel :
         popregs
         ret

spcf_display_unformatted endp

;**********************************************************************
; Set active keys for the dump display; mainly used to fancily
;       turn PgUp and PgDn on and off
;
; Code below assumes the buffer size (and thus offsets) will never
;       approach the 64K limit
;**********************************************************************
set_unformatted_keys    proc    near
        push    bx
        push    ax
        mov     ax, DCJVBP03_KEYS       ;Get active keys
        cmp     pj_offset, 0            ;Are we at top of file?
        jne     chk_setu_1              ;If not, PgUp is OK
        xor     ax, PGUP_OK             ;Else turn off that key
chk_setu_1:
        mov     bx, pj_offset           ;Get present offset
        add     bx, PAGESIZE            ;Add page size
        cmp     bx, pj_datsize          ;Check against data size
        jb      chk_setu_2              ;If less, allow page down
        xor     ax, PGDN_OK             ;Else turn off that key
chk_setu_2:
        mov     Active_Keys, ax
        call    Set_Active_Keys
        pop     ax
        pop     bx
        ret
set_unformatted_keys    endp
```

```
;*************************************************************************
; read_in_file
;
; Reads in file, sets up various parameters
;
;*************************************************************************
read_in_file    proc    near
        mov     filename_ptr, offset PJFILENC   ;Set up file name
        mov     readbuff_ptr, offset Arbbuff    ;Set up buffer
        mov     readbuff_size, pj_bufsize       ;And the buffer size
        call    read_nmvt                       ;Read stuff in
        cmp     read_nmvt_stat, 0               ;Check for success
        jne     dopan_read_notok
;Read was good, so save old file name
        MOVE_STRING PJFILENC,pj_filename_old,PJFILENLV
        jmp     dopan_read_ok
dopan_read_notok:
        mov     pj_datsize, 0                   ;Not good, no data read in
        jmp     read_in_exit                    ;Go do the panel
dopan_read_ok:
        mov     ax,filesize                     ;Get the file size
        mov     pj_datsize, ax                  ;And save the data size
        mov     di, offset PJDMPLNC             ;Get ready to do conversion
        call    Hex2asc                         ;Convert it
read_in_exit:
        ret
read_in_file    endp

restore_filenm  proc near
                MOVE_STRING pj_filename_old,PJFILENC,PJFILENLV
                ret
restore_filenm  endp

PAGE

;**********************************************************************
; format_data takes the data in the buffer and formats it for display
;
; Inputs:
;       Arbbuff         Data buffer
;       pj_offset       Offset into buffer of data to be displayed
;       pj_translate_fg Translation type (ASCII/EBCDIC) (0/1)
;       pj_datsize      Amount of data actually in buffer
;
; Outputs:
;       PJTITLEC        set to correct title line
;       PJDMPHXC        set to hex dump
;       PJDMPCHC        set to character dump
;       PJOFFSEC        set to offset display (field allows entry)
;       PJOFFSTC        set to offset display
;**********************************************************************
format_data     proc    near
;first, calculate the number of lines to be converted, plus the
;       leftovers
;Then, set up title and F key to indicate type of dump (ASCII or EBCDIC)
        cmp     pj_translate_fg, ASCII_FG       ;ASCII translation?
        jne     setup_ebcdic                    ;If not, EBCDIC display
        MOVE_STRING PJTITLE_A,PJTITLEC,PJTITLE_LENGTH   ;show ASCII dump
        MOVE_STRING PJOTYPE_E,PJOTYPEC,PJOTYPELV        ;Other type is EBCDIC
;Set up translation
        mov     pj_translate_ptr, offset ASCII_DISPLAY_TABLE
        jmp     format_next0

setup_ebcdic:
        MOVE_STRING PJTITLE_E,PJTITLEC,PJTITLE_LENGTH   ;show EBCDIC dump
        MOVE_STRING PJOTYPE_A,PJOTYPEC,PJOTYPELV        ;Other type is ASCII
;Set up translation
        mov     pj_translate_ptr, offset EBCDIC_DISPLAY_TABLE
        jmp     format_next0

;Now, do some work
format_next0:
        mov     pj_offset_ptr, offset PJOFFSEC  ;Set up offset array pointer
        mov     pj_dumphx_ptr, offset PJDMPHXC  ;Set up hex dump array pointer
        mov     pj_dumpch_ptr, offset PJDMPCHC  ;Set up char dump array pointer
        mov     ax,pj_offset                    ;Remember the starting
        mov     pj_offset_save,ax               ;       offset
        mov     cx,pj_nlines                    ;Use cx as # of lines counter
        cmp     cx,0
        jle     format_exit     ;If so, just do the rest of the stuff
lineloop:
        push    cx              ;save it
        call    display_line
        pop     cx
        loop    lineloop        ;Loop until done

;************ Now go home
format_exit:
        mov     ax,pj_offset_save       ;Restore original offset
        mov     pj_offset, ax
        ret
format_data     endp

PAGE

;*********************************************************************
; display_line
;
; Takes 16 bytes, calculates offset, ASCII representation of a HEX
;       dump, and an ASCII or EBCDIC character display, placing
;       these in the proper buffers
;
; Input:
;       pj_offset       - offset into buffer to start display at
;       pj_offset_ptr   - pointer into offset output buffer
;       pj_dumphx_ptr   - pointer into hex dump output buffer
;       pj_dumpch_ptr   - pointer into character dump output buffer
;
; Output:
;       PJOFFSEC        - Set to starting offset value (ASCII hex)
;       PJOFFSTC        - Set to appropriate offset values (ASCII hex)
;       PJDMPHXC        - Set to dump values (ASCII hex)
;       PJDMPCHC        - Set to dump values (ASCII or EBCDIC)
;       pj_offset       - offset into buffer to start display at
;       pj_offset_ptr   - pointer into offset output buffer
;       pj_dumphx_ptr   - pointer into hex dump output buffer
;       pj_dumpch_ptr   - pointer into character dump output buffer
;
;**********************************************************************
display_line    proc near
        ;********* Convert the offset for the offset array **********
        mov     ax,pj_offset            ;Get offset value
        mov     di,pj_offset_ptr        ;Set up offset pointer
        call    Hex2asc                 ;Change offset to hex
        add     di,4                    ;bump the pointer
        mov     pj_offset_ptr,di        ;And save the bumped pointer
;********* Convert the buffer values: do hex dump **********
        mov     cx,COLUMNS              ;Set up loop values
        mov     bx,ax                   ;Set bx up to present offset
        mov     di,pj_dumphx_ptr        ;Set up output pointer for hex dump
dump_hx_loop:
        cmp     bx,pj_datsize           ;Past data size?
        jl      still_in                ;No, do standard stuff
        mov     word ptr [di],EMPTY     ;Else put characters indicating
        add     di,2                    ;Update pointer
        jmp     do_next_hex             ;And go do the next stuff
still_in:
        mov     al,offset Arbbuff[bx]   ;Get present value
        call    hexb2asc                ;Convert and output it
        add     di, 2                   ;Bump the pointer
do_next_hex:
        test    cx,1                    ;Are we on an odd count
        je      do_middle_check         ;Check for middle
        mov     byte ptr [di],' '       ;Put out a space every two bytes
        add     di,1                    ;Update the output pointer
do_middle_check:
        cmp     cx,9                    ;We want two spaces in the middle
        jne     dumphx_cont             ;If not, keep going
        mov     byte ptr [di],' '       ;Else get a second space
        add     di,1                    ;And bump the pointer
dumphx_cont:
        add     bx, 1                   ;Increment offset into buffer
        loop    dump_hx_loop            ;Loop until done
        dec     di                      ;Get rid of space at end of line
        mov     pj_dumphx_ptr, di       ;Save the output pointer

;********* Convert the buffer values: do character dump **********
        push    bp
        mov     cx,COLUMNS              ;Set up loop values
        mov     bp,pj_offset            ;Set bx up to present offset
        mov     di,pj_dumpch_ptr        ;Set up output pointer for char dump
        mov     si,pj_translate_ptr     ;Set up translation table pointer
dump_ch_loop:
        cmp     bp, pj_datsize          ;Are we still looking at data?
        jl      dump_ch_ok              ;If so, do real stuff
        mov     byte ptr [di], EMPTYC   ;Else indicate empty
        jmp     dump_ch_cont
dump_ch_ok:
        mov     bl,offset Arbbuff[bp]   ;Get present value
        xor     bh,bh                   ;Zero out high byte
        mov     dh, byte ptr [bx][si]   ;Get translate value for byte 1
        mov     byte ptr [di],dh        ;And move it to the output
dump_ch_cont:
        inc     di                      ;Update the output pointer
        inc     bp                      ;Set bp to next datum
        cmp     cx,9                    ;We want two spaces in the middle
        jne     dumpch_cont             ;If not at middle, keep going
        mov     byte ptr [di],' '       ;Else output a space
        add     di,1                    ;And bump the pointer
dumpch_cont:
        loop    dump_ch_loop            ;Loop until done
        mov     pj_dumpch_ptr, di       ;Save the output pointer
        mov     pj_offset, bp           ;And save the new offset
        pop     bp                      ;Restore base pointer

;********* Clean up and go home      **********
        ret
display_line    endp

PAGE

;**********************************************************************
; setup_offset
;
; Looks at offset field on display and uses it to set pj_offset if it
;       is within bounds (bounds checking not in place yet)
;
; Input:
;       PJOFFSEC - 4 byte ASCII representation of a hexadecimal number
;                       (used in panel display)
;       pj_datsize - Integer representing length of dump (used only for
;                       bounds checking).
; Output:
;       pj_offset - set to desired value (value in PJOFFSEC), if
;                       conversion successful
;       EZ-VU message put up if error found during conversion;
;                       also, cursor moved to field
;**********************************************************************
setup_offset    proc near
        push    di
        push    ax
        mov     di, offset PJOFFSEC     ;Set pointer to characters
        mov     cx, PJOFFSELV           ;Length of PJOFFSEC
        call    asc2hex                 ;Do the conversion
        cmp     cx, 0                   ;Was it successful?
        jne     bad_setup_offset_cnv    ;If no, put up error
        cmp     ax, pj_datsize          ;Was it less than data size?
        jb      setup_offset_ok         ;If so, continue
        mov     ax, PJ_BAD_OFFSET       ;Else put up an error message
        jmp     bad_setup_exit
setup_offset_ok:
        mov     pj_offset, ax           ;Else set up pj_offset
        jmp     exit_setup_offset       ;And exit
bad_setup_offset_cnv:
        mov     ax, PJ_NON_HEX          ;Set error message
bad_setup_exit:
        call    show_errmsg             ;Make EZ-VU display it
        ;Set up field name
        MOVE_STRING PJOFFSEC_PARM, ZFLD, pj_fieldnamel
        DMPC ISPASM,<LNGTH1OPD,PARM1OD,EZVU_RC> ; Reposit cursor
        mov     cx,1                    ;Make sure cx is nonzero
exit_setup_offset:
        pop     ax
        pop     di
        ret
setup_offset endp

;*********************************************************************
; do_pgup
;
; Do a page up on dump (unformatted) panel
;
; Input:
;       pj_offset       - present offset into buffer
;
; Output:
;       pj_offset       - new offset into buffer
;
; Modifies ax, also may output error message
;
;*********************************************************************
do_pgup proc    near
        mov     ax, PAGESIZE                    ;Get display page
        cmp     ax, pj_offset                   ;Is it bigger than page size?
        jle     pg_up_setup                     ;No, so set it up
        mov     ax, PJ_BEGINNING                ;Set up error code
        call    show_errmsg                     ;And show error message
        mov     ax, pj_offset                   ;Amount to subtract is any
                                                ;       non-zero offset amount
pg_up_setup:
        sub     pj_offset,ax                    ;Do the actual subtraction
        ret
do_pgup endp

;*********************************************************************
; do_pgdn
;
; Do a page down on dump (unformatted) panel
;
; Input:
;       pj_offset       - present offset into buffer
;       pj_datsize      - length of data
;
; Output:
;       pj_offset       - new offset into buffer
;
; Modifies ax, also may output error message
;
;*********************************************************************
do_pgdn proc    near
        mov     ax, PAGESIZE                    ;Get PAGE SIZE
        sub     ax, COLUMNS                     ;Page down NLINES-1 lines
        add     ax, pj_offset                   ;Get the new offset
        cmp     ax, pj_datsize                  ;Is it bigger than page size?
        jl      pg_dn_setup                     ;No, so go set it up
        mov     ax, PJ_ENDING                   ;Else set up error code
        call    show_errmsg                     ;And show error message
;       mov     ax, pj_datsize                  ;Amount to subtract is any
                                                ;       non-zero offset amount
;       sub     ax,PAGESIZE                     ;Display last page
        mov     ax,pj_offset                    ;If we would go off end of
                                                ;       data, don't move
pg_dn_setup:
        mov     pj_offset,ax                    ;Set up new offset
        ret
do_pgdn endp


;*************************************************************************
; spcf_display_formatted
;
; This is the panel handler for the formatted display.
;
; Input:
;       PJFILENC        - file to be displayed
;
; Output:
;       PJFILENC        - may be modified by user
;
;*************************************************************************
PUBLIC spcf_display_formatted
spcf_display_formatted  proc near
        pushregs
dofpanel_read:
        mov     filename_ptr, offset PJFILENC   ;Set up file name
        mov     readbuff_ptr, offset Arbbuff    ;Set up buffer
        mov     readbuff_size, pj_bufsize       ;And the buffer size
        call    read_nmvt                       ;Read stuff in
        cmp     read_nmvt_stat, 0               ;Check for success
        MOVE_STRING PJFILENC, pj_filename_old, PJFILENLV
        jne     dofpan_read_notok
        jmp     dofpan_read_ok

dofpan_read_notok:
        mov     pj_datsize, 0                   ;Not good, no data read in
        call    format_nothing                  ;Clear out data areas
        jmp     dofpanel_ok                     ;And display blanks

dofpan_read_ok:
        mov     ax,filesize                     ;Get the file size
        mov     pj_datsize, ax                  ;And save the data size

dofpanel_format:
         mov    pj_offset, 0                    ;Set initial offset to 0
         call   format_formatted_data           ;Format the data
         cmp    ax,0                            ;If not OK, display unf.
         je     dofpanel_ok                     ;Else display panel

         cmp     pj_cross, 0                    ;Did we come from unformatted?
         jne     formatted_cross1               ;If non-zero, we did
         call    spcf_display_unformatted       ;Else go do unformatted dump,

         cmp     Zrsp1,F3                       ;Was F3 the exit key from
                                                ;the unformatted display panel?
         jne     back_to_formatted              ;No, returned to this panel.
         jmp     exit_dofpanel                  ;Yes, wanted to quit so exit

back_to_formatted:
         jmp     dofpanel_read                  ;redisplay unformatted panel

formatted_cross1:                               ;We came from unformatted, so
        mov     pj_cross, 0                     ;       clear the flag and
        jmp     exit_dofpanel                   ;       exit this panel

dofpanel_ok:
         mov     Active_Keys, DCJVBP04_KEYS      ;Set active keys
         call    Set_Active_Keys
         DMPC    ISPASM,<DCJVBP04L,DCJVBP04,EZVU_RC>

display_dofpanel:
         cmp     Zrspl,F3                       ;was f3 the exit key?
         jne     not_f3_3f
         call    restore_filenm                 ;Restore old file name
         mov     pj_cross, 0                    ;Clear out crossing flag
         jmp     exit_dofpanel                  ;yes, exit this rtn

not_f3_3f:
         cmp     Zrspl,F6                       ;was it dos request?
         jne     not_f6_3f
         call    restore_filenm                 ;Restore old file name
         call   execpgm                         ;Invoke second command
                                                ;      processor
         jmp     dofpanel_read                  ;On exit, re-read file
                                                ; and redisplay

not_f6_3f:
         CMP     Zrspl,F12                      ;Was it display unformatted?
         jne     not_f12_3f                     ;If not, check next
         call    restore_filenm                 ;Restore old file name
         cmp     pj_cross, 0                    ;Did we come from unformatted?
         jne     formatted_cross                ;If non-zero, we did
         mov     pj_cross, 1                    ;Otherwise, we didn't, so set
         call    spcf_display_unformatted       ;       the flag and call

         cmp     Zrspl,F3                       ;Was F3 the exit key from
                                                ;the unformatted display panel?

         je      jmp_to_exit_dofpanel           ;Yes exit proc
         jmp     dofpanel_read                  ;No, switch to formatted
                                                ;    redisplaying on return
jmp_to_exit_dofpanel:
         jmp     exit_dofpanel

formatted_cross:                                ;We came from unformatted, so
        mov     pj_cross, 0                     ;       clear the flag and
        jmp     exit_dofpanel                   ;       exit this panel

not_f12_3f:
        cmp     Zrspl,F9                        ;Was it an F9?
        jne     redisplay_dofpanel              ;If not, redisplay panel
        jmp     dofpanel_read                   ;Read in new data
no_fn_change:
        jmp     redisplay_dofpanel              ;Then redisplay

not_F9_3f:

refresh_dofpanel:
        call    format_formatted_data           ;Do new data
redisplay_dofpanel:
        DMPC ISPASM,<LNGTH8PD,PARM8D,EZVU_RC>   ;Else redisplay
        jmp     display_dofpanel                ;And redisplay panel
```

```
exit_dofpanel:
        popregs
        ret
spcf_display_formatted  endp

;*********************************************************************
;format_formatted_data:
;
; Takes the data in the input file, checks it to make sure it is a
;       recognized NMVT.  If so, it formats all data for display.
;       If not, returns error in AX and sets up an EZ-VU error message.
;
; Input:
;       pj_arb          - contains data read in from file
;
; Output:
;
;       ax              - Zero if no error, error number if error
;       *               - All data for panel DCJVBP04 set up
;*********************************************************************

format_formatted_data   proc    near
;First, check the ARB ID
        COMPARE_STRINGS pj_arb.pj_arbid, P04ARBID, P04ARBIDLV
        je      fcheck_reqcode          ;If ok, check request code
        mov     ax, P04_BAD_ARBID       ;Else set error message
        jmp     ffd_bad
fcheck_reqcode:
        cmp     pj_arb.pj_reqcode, P04_request_code  ;Check request
        je      fcheck_arblen           ;If ok, check length
        mov     ax, P04_BAD_REQCODE     ;Else set up error message
        jmp     ffd_bad
fcheck_arblen:
        mov     al, pj_arb.pj_arblen    ;Get arb length
        cmp     al, P04ARBLN            ;And check it
        je      fcheck_parseid          ;OK, check the parse ID
        mov     ax, P04_BAD_ARBLEN      ;No good, set up error message
        jmp     ffd_bad
fcheck_parseid:
        mov     al,pj_arb.pj_parseid    ;Get parse id
        sub     ah, ah                  ;Clear out top half
        sub     al, P04_MIN_PID         ;Make sure it is in bounds
        jl      fbad_parseid
        cmp     al, P04_NUM_PID         ;Was over max, make sure it is under
                                        ;maximum number of procs
        jge     fbad_parseid            ;If over or equal, was no good
        inc     ax                      ;Add one to put us past dummy in the
                                        ;       dispatch table
        rol     ax,1                    ;Multiply by 2 to get indexes
        push    ax                      ;Save it
        cmp     p04_jump_offset, 0      ;Is the offset presently 0?
        je      offset_ok               ;If so, continue
        mov     bx, p04_jump_offset     ;Else delete leftover variables
        call    word ptr p04_vdel_dispatch[bx]
offset_ok:
        pop     ax                      ;Restore offset
        mov     p04_jump_offset,ax      ;And store the offset
        call    do_real_format
        jmp     ffd_good                ;Return A-OK sign

fbad_parseid:
        mov     ax, P04_BAD_PARSEID     ;Indicate bad parse ID
        jmp     ffd_bad
ffd_bad:
        call    show_errmsg     ;Display error message
        jmp     ffd_exit
ffd_good:
        mov     ax,0            ;Indicate no error
        jmp     ffd_exit
ffd_exit:
        ret
format_formatted_data   endp

;**********************************************************************
; The following is the code that does setup of common variables as well
;       as dispatching for vdefines and the resource/command fields
;**********************************************************************
do_real_format  proc    near
        mov     ax, pj_arb.pj_retcode   ;Get return code
        mov     P04RETCD, ax            ;Store it in EZ-VU variable
        mov     ax, pj_arb.pj_errclass  ;Get error class
        mov     P04ERCLS, ax
        mov     ax, pj_arb.pj_errtype   ;Get error type
        mov     P04ERTYP, ax
        mov     di, offset P04PRSNS     ;Now do parse sense - requires
        mov     bx, offset pj_arb.pj_sensedata  ;conversion to hex
        mov     cx, P04_SENSE_LEN
p4_sen_loop:                            ;Convert the parse sense stuff
        push    cx
        mov     al, byte ptr [bx]
        call    Hexb2asc
        add     di, 2
        inc     bx
        pop     cx
        loop    p4_sen_loop

        mov     al, pj_arb.pj_parseid   ;Set up parse ID
        mov     di, offset P04PRSID     ;It is hex, so set up conversion output
        call    Hexb2asc
;Set up the receive correlator
        mov     cx,P04RCVCRLV           ; Set CX to 10 bytes
        ror     cx,1
        mov     di,offset P04RCVCR      ; Point DI at Hex/ASCII string buffer
        mov     si,offset pj_arb.pj_recvcorr ; Point SI at Receive correlator
setup_rcv_loop:
        mov     al, byte ptr [si]
        call    hexb2asc                 ;Convert the bytes
        inc     si                      ;Bump binary pointer
        add     di, 2                   ;Bump output pointer
        loop    setup_rcv_loop          ;And loop until done

        mov     bx, p04_jump_offset     ;Get ready to do vdefines
        push    bx                      ;Save the offset


;********* Do Vdefines for this ARB type
```

```
        call    word ptr p04_vdef_dispatch[bx]
        pop     bx                      ;Restore offset
        push    bx
        FILL_CHAR P04RDATA,' ',P04RDATALV     ;Erase data area


;********* Set up the title line
        pop     bx
        mov     cx, PJTITLE_LENGTH
        mov     si, word ptr p04_titles[bx]
        mov     di, offset PJTITLEC
        push    ds
        pop     es
        cld
        rep     movsb


;********* Call formatting procedure for this ARB type
        call    word ptr p04_form_dispatch[bx]
        ret
do_real_format  endp

;**********************************************************************
; The following is the code that clears things in the event an attempt
;       is made to read an unknown file
;**********************************************************************
format_nothing  proc    near
        pushregs
        mov     bx, p04_jump_offset     ;Get jump offset
        cmp     bx, 0                   ;Is the offset presently 0?
        je      offset_ok_null          ;If so, continue
        call    word ptr p04_vdel_dispatch[bx]
offset_ok_null:
        mov     p04_jump_offset,0       ;And zero it out
        mov     ax, -1
        mov     P04RETCD, ax            ;Store it in EZ-VU variable
        mov     ax, -1
        mov     P04ERCLS, ax
        mov     ax, -1
        mov     P04ERTYP, ax
        mov     al, 0                   ;Set up parse ID
        mov     P04PRSID, al
        mov     P04CMDLN, al            ;Also zero out command count
        mov     di, offset P04PRSNS     ;Now blank out parse sense
        mov     al, SPACEC
        mov     cx, P04PRSNSLV          ;Length
        push    ds
        pop     es
delete_prsns_loop:
        mov     byte ptr [di], al
        inc     di                      ;Bump output pointer
        loop    delete_prsns_loop       ;And loop until done

;Set up the receive correlator
        mov     di,offset P04RCVCR     ; Point DI at Hex/ASCII string buffer
        mov     cx, P04RCVCRLV         ;Length
        mov     al, SPACEC
        push    ds
        pop     es
delete_rcv_loop:
        mov     byte ptr [di], al
        inc     di                      ;Bump output pointer
        loop    delete_rcv_loop          ;And loop until done
        FILL_CHAR P04RDATA,SPACEC,P04RDATALV ;Erase data area
        MOVE_STRING P04CMDTX_Null,P04CMDTX,P04CMDTXLV

;********* Set up the title line
        mov     bx, 0                   ;Index 0 is empty stuff
        mov     cx, PJTITLE_LENGTH
        mov     si, word ptr p04_titles[bx]
        mov     di, offset PJTITLEC
        push    ds
        pop     es
        cld
        rep     movsb
        popregs
        ret

format_nothing  endp


;***********************************************************************
; format_run:
;
; Set up data fields to display the RUN command
;***********************************************************************
format_run      proc near
        pushregs
        push    ds
        pop     es
        mov     al, pj_arb.pj_commandlen        ;Save command length
        sub     ah, ah
        mov     P04CMDLN, al
        push    ax
        MOVE_STRING P04_COLLINE, P04RDATA, P04_COLUMNS
        MOVE_STRING P04CMDTX_Command,P04CMDTX,P04CMDTXLV
        pop     ax
        mov     cx, ax
;Line below assumes command immediately follows data
;        mov     si, offset pj_arb.pj_parsedata
;But we want to be good, and use the offset portion of the command pointer
; in the ARB
        mov     si, word ptr pj_arb.pj_command  ;Get offset into si
        add     si, offset pj_arb               ;add buffer offset value
        mov     di, offset P04RDATA2
        push    ds
        pop     es
        cld
        rep     movsb
        popregs
        ret
format_run      endp


;***********************************************************************
; format_link:
;
```

```
; Set up data fields to display the various link commands
;*********************************************************************
format_link     proc near
        pushregs
        ;Set up Number of resources field
        MOVE_STRING P04CMDTX_Resources, P04CMDTX, P04CMDTXLV
        mov     ax, pj_arb.pj_testcount         ;Store test count
        mov     P04TSCNT, ax
        mov     al, pj_arb.pj_testtype          ;And test type
        mov     P04TSTYP, al
        mov     al, pj_arb.pj_numnames          ;And number of names
        mov     P04CMDLN, al
        cmp     ax, 0                           ;More than zero names?
        jne     move_names                      ;If so, move them in
        jmp     format_link_exit                ;Else exit
move_names:
        mov     cx, ax                          ;Now let's put it in RDATA
        mov     di, offset P04RDATA             ;Set output address
;        mov     si, offset pj_arb.pj_parsedata
        mov     si, word ptr pj_arb.pj_names    ;Get offset of names into si
        add     si, offset pj_arb               ;add buffer offset value
        push    ds
        pop     es
rnames_loop:
        push    cx                              ;Save loop counter
        push    di                              ;And present output address
        mov     cl, byte ptr [si]               ;Get length of name
        sub     ch,ch
        inc     si                              ;Point to name itself
        cld
        rep     movsb                           ;Move the name
        pop     di                              ;Restore old output
        add     di, 9                           ;Move to next slot
        pop     cx                              ;Get back loop counter
        loop    rnames_loop                    ;And loop until done

format_link_exit:
        popregs
        ret
format_link     endp

;*********************************************************************
;define_link_test performs a VDEFINE on the five fields needed for the
;       LINK TEST display
;*********************************************************************
define_link_test     proc    near
       DMPC_NS ISPASMV,<P04RESRCLP,P04RESRC_PARM,EZVU_RC,P04RESRC,P04RESRCLV>
       DMPC_NS ISPASMV,<P04TSCNXLP,P04TSCNX_PARM,EZVU_RC,P04TSCNX,P04TSCNXLV>
       DMPC_NS ISPASMV,<P04TSCNTLP,P04TSCNT_PARM,EZVU_RC,P04TSCNT,P04TSCNTLV>
       DMPC_NS ISPASMV,<P04TSTYXLP,P04TSTYX_PARM,EZVU_RC,P04TSTYX,P04TSTYXLV>
       DMPC_NS ISPASMV,<P04TSTYPLP,P04TSTYP_PARM,EZVU_RC,P04TSTYP,P04TSTYPLV>
                ret
define_link_test     endp

;*********************************************************************
;delete_link_test performs a VDELETE on the five fields needed for the
;       LINK TEST display
;*********************************************************************
delete_link_test     proc    near
```

```
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04RESRC_DELETE, EZVU_RC>
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04TSCNX_DELETE, EZVU_RC>
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04TSCNT_DELETE, EZVU_RC>
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04TSTYX_DELETE, EZVU_RC>
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04TSTYP_DELETE, EZVU_RC>
                ret
delete_link_test     endp

;*********************************************************************
;define_link_data performs a VDEFINE on the fields needed for the
;       LINK DATA display
;*********************************************************************
define_link_data     proc    near
       DMPC_NS ISPASMV,<P04RESRCLP,P04RESRC_PARM,EZVU_RC,P04RESRC,P04RESRCLV>
                ret
define_link_data        endp

;*********************************************************************
;delete_link_data performs a VDELETE on the fields needed for the
;       LINK DATA display
;*********************************************************************
delete_link_data     proc    near
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04RESRC_DELETE, EZVU_RC>
                ret
delete_link_data     endp


;*********************************************************************
;define_link_pd performs a VDEFINE on the fields needed for the
;       LINK PD   display
;*********************************************************************
define_link_pd     proc    near
       DMPC_NS ISPASMV,<P04RESRCLP,P04RESRC_PARM,EZVU_RC,P04RESRC,P04RESRCLV>
                ret
define_link_pd        endp

;*********************************************************************
;delete_link_pd performs a VDELETE on the fields needed for the
;       LINK DATA display
;*********************************************************************
delete_link_pd       proc    near
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04RESRC_DELETE, EZVU_RC>
                ret
delete_link_pd       endp

;*********************************************************************
;define_run performs a VDEFINE on the fields needed for the
;       LINK DATA display
;*********************************************************************
define_run     proc    near
       DMPC_NS ISPASMV,<P04CMDLILP,P04CMDLI_PARM,EZVU_RC,P04CMDLI,P04CMDLILV>
                ret
define_run        endp

;*********************************************************************
;delete_run performs a VDELETE on the fields needed for the
;       RUN display
;*********************************************************************
delete_run       proc    near
```

```
        DMPC_NS ISPASM,<P04_DELETE_LEN, P04CMDLI_DELETE, EZVU_RC>
                ret
delete_run       endp

CSEG    ENDS


end
```

Appendix J. NetView Sample Programs

NetView Sample Presentation Services Command Processor (PSCP)

```
         TITLE  'COPYRIGHT INTERNATIONAL BUSINESS MACHINES CORPORATION'
* ***************************************************************
*        API Sample Program - (C) Copyright IBM Corp. 1986, 1987
*        SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
* You are hereby licensed to use, reproduce, and distribute these sample
* programs as your needs require.   IBM does not warrant the suitability
* or integrity of these sample programs and accepts no responsibility for
* their use for your applications.  If you choose to copy and redistribute
* significant portions of these sample programs, you should preface such
* copies with this copyright notice.
* ***************************************************************
*
      PRINT NOGEN
 DSICBS  DSICBH,DSIPDB,DSISWB,DSITIB,DSITVB,DSIMVT,DSISVL,DSIIFR,DSICWB
DSINVPCP CSECT
         USING *,R15
         B     SAVEREGS
CSECTNAM DC    CL10'DSINVPCP'
         DC    CL42'COPYRIGHT INTERNATIONAL BUSINESS MACHINES'
         DC    CL18'CORPORATION, 1986,1987'
         DC    C'&SYSDATE'
      PRINT GEN
* ***************************************************************
*
* INPUT INTO THIS COMMAND PROCESSOR IS A BUFFER CONTAINING THE TEXT
* OF A MESSAGE FROM THE NETVIEW OPERATOR.
* NORMAL OUTPUT IS AN IFR SENT TO DSCP NAMED DSISPCFD.
* SEVERAL MESSAGES ARE PRINTED TO THE OPERATOR IF THE INPUT TEXT
* IS NOT IN THE REQUIRED ORDER, HAS IN UNACCEPTABLE LENGTH, OR IF
* KEYWORDS ARE MISSING OR ARE SPELLED INCORRECTLY.
*
* ***************************************************************
R15      EQU  15
R14      EQU  14
R13      EQU  13
R12      EQU  12
R11      EQU  11
R10      EQU  10
R9       EQU   9
R8       EQU   8
R7       EQU   7
R6       EQU   6
R5       EQU   5
R4       EQU   4
R3       EQU   3
R2       EQU   2
R1       EQU   1
R0       EQU   0
*
SAVEREGS EQU   *
* ***************************************************************
* REGISTER SAVE CONVENTIONS START HERE
```

```
*
         DROP  R15
         STM   R14,R12,12(R13)
         LR    R12,R15
         USING DSINVPCP,R12           MODULE ADDRESSABILITY
         LR    R9,R1                  BASE FOR CWB
         USING DSICWB,R9
         LA    R2,CWBSAVEA            GET MY SAVERAREA ADDR
         ST    R13,4(R2)              BACKWARD CHAIN
         ST    R2,8(R13)              FORWARD POINTER
         LR    R13,R2                 THIS SAVEA IN R13
         XC    8(4,R13),8(R13)        ZERO FORWARD POINTER
         L     R10,CWBTIB             GET TIB ADDRESS
         USING DSITIB,R10
         L     R6,TIBTVB              GET TVB ADDRESS
*    SAVE CONVENTIONS COMPLETE
*
         L     R3,CWBPDB              PDB ADDRESS
         USING DSIPDB,R3              BASE FOR PDB
         LA    R8,CWBADATD            POINT TO THE CWB BUFFER
         USING AUTOWORK,R8            BASE FOR CWB WORK BUFFER
         MVI   AUTOWORK,X'00'         ZERO FIRST BYTE
         MVC   AUTOWORK+1(255),AUTOWORK  ZERO THE REMAINDER
* *************************************************************
*    REGISTERS CURRENTLY SET UP
*
*          USING       R3 IS PDB BASE
*          USING       R8 IS BASE FOR CWBADATD BUFFER
*          USING       R9 IS CWB ADDRESS
*          USING      R10 IS TIB ADDRESS
*          USING      R12 IS MODULE BASE
* *************************************************************

         MVI   ERRINDC,X'00'          INITIALIZE CWB ERROR INDICATOR
         MVC   RETCODE,=F'0'          SET THE RETURN CODE TO ZERO
         BAL   R14,SETUP1             GO SET UP THE OPER OUT MSG BUF
         BAL   R14,INVOKER            CK INVOKER IS OPER OR C-LIST
         BAL   R14,TERMINPT           OK - NOW CHECK THE INPUT MSG
         LH    R6,IFRBLENG            OK - SET UP THE BUFFER SIZE
         STH   R6,GETBLENG              FOR THE IFR BUFFER
         BAL   R14,GETDBUF            OK - GET THE DSCP MSG BUF
         L     R6,GOTNBADR            GET ADDRESS OF BUFFER
         ST    R6,BLDIFRAD            ADDR OF BUFFER TO BLD IFR
         BAL   R14,BLDIFR             OK - BUILD IFR HDR, MOVE TEMPLET
         BAL   R14,MOVDMSG            OK - MOVE THE MSG TO DSCP BUF
         BAL   R14,SENDSCP            OK - SEND OP MSG TO THE DSCP
         LTR   R15,R15                CHECK THE RETURN CODE
         BZ    RESTOR                 OK - THE MESSAGE IS ON ITS WAY
         STC   R15,MQSERRC            NO - SAVE DSIMQS ERROR RC
         L     R11,GOTNBADR           GET ADDR OF BUFFER TO FREE
         ST    R11,FREMADDR           ADDR OF DSIFRE TO FREE
         BAL   R14,FREDBUF            OK - FREE THE DSCP MSG BUFFER
         B     MQSERR                 DISPLAY ERROR MSG AND EXIT
*
* ****************************************
*
INVOKER  EQU   *
         ST    R14,R14SAVE             SAVE THE RETURN REGISTER
* WHY WAS THIS CP INVOKED
```

```
         L     R6,CWBBUF                R6 POINTS TO THE INPUT BUFFER
         USING BUFHDR,R6
         CLI   HDRMTYPE,HDRTYPEC        IS INPUT FROM A CLIST ?
         BE    INPUTOK                  NOT THE OPERATOR
         CLI   HDRMTYPE,HDRTYPET        IS INPUT FROM A TERMINAL ?
         BNE   NOTOPER                  NOT THE OPERATOR
         DROP  R6
INPUTOK  EQU   *                        INPUT IS FROM CLIST OR OPERATOR
         L     R14,R14SAVE             RESTORE THE RETURN REGISTER
         BR   14                        OP OUT MSG BUF SET UP NOW
*
* ****************************************
*
* SET UP THE OPERATOR OUTPUT MESSAGE BUFFER
SETUP1   EQU   *
         ST    R14,R14SAVE              SAVE THE RETURN REGISTER
*  COPY THE HEADER FROM THE COMMAND BUFFER TO THE OPERATOR MSG BUFFER
         LR    R4,R8                    POINT TO WHERE IT GOES
         L     R5,CWBBUF                POINT TO WHERE IT IS
         LA    R2,BUFHDRND-BUFHDR       HOW LONG IT IS
         LR    R6,R8
         USING BUFHDR,R6
         BCTR  R2,0                     -1 FOR THE MOVE
         EX    R2,MOVE                  MOVE THE HEADER
         MVI   HDRMTYPE,HDRTYPEU
         MVI   HDRIND,X'00'
         XC    HDRTSTMP(4),HDRTSTMP
         MVI   HDRTSTMP+3,X'0C'
         LA    R2,BUFHDRND-BUFHDR       HOW LONG IT IS
         STH   R2,HDRTDISP
         DROP  R6
         L     R14,R14SAVE             RESTORE THE RETURN REGISTER
         BR   14                        OP OUT MSG BUF SET UP NOW
*
* ****************************************
*
TERMINPT EQU   *
         ST    R14,R14SAVE             SAVE THE RETURN REGISTER
         SR    R2,R2                   ZERO R2
         LH    R2,PDBNOENT             # OF ENTRIES
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
* FIND THE 'SP = ... ,'
         LA    R7,PDBTABLE             ADDR OF PDB TABLE ENTRY 1
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
*  R7 POINTS TO THE COMMAND NAME ENTRY
         USING PDBENTRY,R7             BASE FOR PDBENTRY
         LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF PDB ENTRY 2
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
*  R7 POINTS TO WHERE THE 'SP' ENTRY SHOULD BE
         L     R6,CWBBUF
*  R6 POINTS TO THE INPUT BUFFER
         AH    R6,PDBDISP              DISPLACEMENT TO 'SP'
         CLC   0(2,R6),SP              IS IT SP?
         BNE   CMDSPERR                SP NOT CORRECT
```

```
*  NOW CHECK FOR THE '='
         CLI   PDBTYPE,X'7E'           IS THERE AN =
         BE    SPEQOK2
         LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDBENTRY
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
         CLI   PDBLENG,X'00'           ZERO LENGTH ENTRY?
         BNE   CMDSPERR                SP NOT CORRECT
         CLI   PDBTYPE,X'7E'           IS THERE AN =
         BE    SPEQOK2
         B     CMDSPERR                SP NOT CORRECT
SPEQOK2  LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDBENTRY
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
*  R7 POINTS TO WHERE THE SPNAME SHOULD BE
* CHECK THAT THE SP NAME IS 8 CHARACTERS OR LESS IN LENGTH
         CLI   PDBLENG,X'08'           SP NAME 8 OR LESS?
         BH    SPNAMER                 NAME IS TOO LONG
         SR    R4,R4                   ZERO R4
         IC    R4,PDBLENG              GET LENGTH OF SP NAME
         LTR   R4,R4                   IF IT IS ZERO
         BZ    SPNAMER                 NAME IS TOO SHORT
         STC   R4,SPNLENG              SAVE THE LENGTH
         L     R6,CWBBUF               R6 POINTS TO THE INPUT BUFFER
         AH    R6,PDBDISP              DISPLACEMENT TO 'SPNAME'
         ST    R6,SPNPTR               SAVE THE POINTER TO SP NAME
         CLI   PDBTYPE,X'6B'           IS THERE A ,
         BE    NOWAPPL                 OK - CHECK THE APPL
         CLI   PDBTYPE,X'40'           IS THERE A BLANK
         BNE   CMDSPERR                NO - SP NOT CORRECT
         LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDB ENTRY
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
         CLI   PDBTYPE,X'6B'           IS THERE A ,
         BNE   CMDSPERR                NO - SP NOT CORRECT
*
NOWAPPL  LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDB ENTRY
*  R7 POINTS TO WHERE THE 'APPL' ENTRY SHOULD BE
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
*  R7 POINTS TO WHERE THE 'APPL' ENTRY SHOULD BE
         L     R6,CWBBUF               R6 POINTS TO THE INPUT BUFFER
         AH    R6,PDBDISP              DISPLACEMENT TO 'APPL'
         CLC   0(4,R6),AP              IS IT APPL?
         BNE   CMDAPERR                APPL NOT CORRECT
*  NOW CHECK FOR THE '='
         CLI   PDBTYPE,X'7E'           IS THERE AN =
         BE    APEQOK2
         LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDBENTRY
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
         CLI   PDBLENG,X'00'           ZERO LENGTH ENTRY?
         BNE   CMDSPERR                SP NOT CORRECT
         CLI   PDBTYPE,X'7E'           IS THERE AN =
         BE    APEQOK2
         B     CMDSPERR                SP NOT CORRECT
APEQOK2  LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDBENTRY
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
*  R7 POINTS TO WHERE THE APPLNAME SHOULD BE
* CHECK THAT THE AP NAME IS 8 CHARACTERS OR LESS IN LENGTH
         CLI   PDBLENG,X'08'           AP NAME 8 OR LESS?
         BH    APNAMER                 NAME IS TOO LONG
         SR    R4,R4                   ZERO R4
         IC    R4,PDBLENG              GET LENGTH OF SP NAME
         LTR   R4,R4                   IF IT IS ZERO
         BZ    APNAMER                 NAME IS TOO SHORT
         STC   R4,APNLENG              SAVE THE LENGTH
         L     R6,CWBBUF               R6 POINTS TO THE INPUT BUFFER
         AH    R6,PDBDISP              DISPLACEMENT TO 'APNAME'
         ST    R6,APNPTR               SAVE THE POINTER TO AP NAME
         CLI   PDBTYPE,X'6B'           IS THERE A ,
         BE    NOWTEXT                 OK - CHECK THE TEXT
         CLI   PDBTYPE,X'40'           IS THERE A BLANK
         BNE   CMDAPERR                NO - APPL NOT CORRECT
         LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDB ENTRY
         BCTR  R2,0                    DECREMENT BY 1
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    CMDERR                  NOT ENOUGH ENTRIES
         CLI   PDBTYPE,X'6B'           IS THERE A ,
         BNE   CMDAPERR                NO - APPL NOT CORRECT
         CLI   PDBLENG,X'00'           ZERO LENGTH?
         BNE   CMDSPERR                NO - APPL NOT CORRECT
*
NOWTEXT  LA    R7,PDBENTND-PDBENTRY(,R7) ADDR OF NEXT PDB ENTRY
* SAVE THE TEXT MESSAGE POINTER
         LTR   R2,R2                   HOW MANY ARE THERE
         BZ    NOCMD                   NOT ENOUGH ENTRIES
         L     R6,CWBBUF               R6 POINTS TO THE INPUT BUFFER
         AH    R6,PDBDISP              DISPLACEMENT TO TEXT
         ST    R6,TXTPTR               SAVE THE POINTER TO TEXT START
* SAVE THE TEXT MESSAGE LENGTH
         L     R6,CWBBUF               R6 POINTS TO THE INPUT BUFFER
         USING BUFHDR,R6
         SR    R2,R2                   ZERO R2
         SR    R4,R4                   ZERO R4
         LH    R4,HDRMLENG             GET THE MESSAGE LENGTH
         LH    R2,PDBDISP              GET THE OFFSET TO TEXT START
         STC   R2,TXTDISP              SAVE THE OFFSET TO THE TEXT
         SH    R2,HDRTDISP             SUBTRACT OFFSET TO 1RST CHARAC.
         SR    R4,R2                   LENGTH OF TEXT IN R4
         STC   R4,TXTLENG              SAVE THE LENGTH
         SR    R2,R2                   ZERO R2
         LA    R2,80
         CLR   R4,R2                   CHECK LENGTH OF TEXT
         BH    NOCMD                   CMD TEXT IS TOO LONG
         LA    R4,(BUFHDRND-BUFHDR)(R4) ADD THE LENGTH OF A BUFHDR
         LA    R4,(TEXT-IFRBUFR)(R4)   ADD LGTH UP TO THE TEXT
         STH   R4,IFRBLENG             SAVE THE LENGTH TO GET
         DROP  R3
         DROP  R6
         L     R14,R14SAVE             RESTORE THE RETURN REGISTER
         BR    R14                    DSCP MESSAGE IS IN THE BUFFER
*
* *****************************************
*
MOVDMSG  EQU   *
         ST    R14,R14SAVE             SAVE THE RETURN REGISTER
         L     R11,BLDIFRAD
         USING BUFHDR,R11              BASE THE GOTTEN IFR BUFFER
* NOW FILL IN THE SP NAME, APPL NAME, AND MESSAGE TEXT
         LA    R6,BUFHDRND            POINT TO THE IFR MESSAGE AREA
         DROP R11
         USING IFRBUFR,R6
         LA    R4,SPNAME              R4 POINTS TO WHERE IT GOES
         L     R5,SPNPTR              R5 POINTS TO WHERE IT IS
         SR    R2,R2                  ZERO R2
         IC    R2,SPNLENG             R2 HAS LENGTH TO MOVE
         BCTR  R2,0
         EX    R2,MOVE                MOVE THE SP NAME
         LA    R4,APNAME              R4 POINTS TO WHERE IT GOES
         L     R5,APNPTR              R5 POINTS TO WHERE IT IS
         SR    R2,R2                  ZERO R2
         IC    R2,APNLENG             R2 HAS LENGTH TO MOVE
         BCTR  R2,0
         EX    R2,MOVE                MOVE THE APPL NAME
         LA    R4,TEXT                R4 POINTS TO WHERE IT GOES
         L     R5,TXTPTR              R5 POINTS TO WHERE IT IS
         SR    R2,R2                  ZERO R2
         IC    R2,TXTLENG             R2 HAS THE LENGTH TO MOVE
         BCTR  R2,0
         EX    R2,MOVE                MOVE THE TEXT
         LA    R2,2(,R2)              ADD TWO FOR LENGTH
         STC   R2,TXTLNGTH            STORE TEXT LENGTH IN IFR BFR
         L     R14,R14SAVE            RESTORE THE RETURN REGISTER
         BR    R14                    THE DSCP MSG BUFFER IS SET UP
         DROP  R6
*
* *******************************************************************
* BUILD A BUFHDR AND MOVE THE IFR TEMPLET INTO THE BUFFER AT BLDIFRAD
BLDIFR   ST    R14,R14SAVE             SAVE THE RETURN REGISTER
         L     R11,BLDIFRAD
         USING BUFHDR,R11
*  COPY THE HEADER FROM THE COMMAND BUFFER TO THE DSCP MSG BUFFER
         LR    R4,R11                  POINT TO WHERE IT GOES
         L     R5,CWBBUF               POINT TO WHERE IT IS
         LA    R2,BUFHDRND-BUFHDR      HOW LONG IT IS
         BCTR  R2,0                    -1 FOR THE MOVE
         EX    R2,MOVE                 MOVE THE HEADER
* ***********************************************************
*    REGISTERS CURRENTLY SET UP
*           USING       R8 IS BASE FOR DSRBUSER
*           USING       R9 IS CWB ADDRESS
*           USING      R10 IS TIB ADDRESS
*           USING      R11 IS BUFHDR AT GOTTEN BUFFER ADDRESS
*           USING      R12 IS MODULE BASE
* ***********************************************************
* BUILD THE IFR BUFFER HEADER IN THE NEW BUFFER
         LA    R2,BUFHDRND-BUFHDR     LENGTH OF BUF HEADER
         STH   R2,HDRTDISP            STORE OFFSET TO MSG
         LH    R6,IFRBLENG            GET BUFFER LENGTH
         STH   R6,HDRBLENG            BUF LENGTH IN HEADER
         SLR   R6,R2
         STH   R6,HDRMLENG            MESSAGE LENGTH
         MVI   HDRMTYPE,HDRTYPEI      INTERNAL FUNCTION
*
         L     R7,TIBTVB              ADDR OF TVB
         USING DSITVB,R7              BASE FOR TVB
         L     R4,TVBMVT              GET MY MVT ADDR
         USING DSIMVT,R4              BASE FOR MVT
*
         MVC   HDRDOMID,MVTCURAN
         DROP  R4
         DROP  R7
* FILL IN THE IFR BUFFER TEMPLET
         LA    R4,BUFHDRND            POINT TO IFR MESSAGE AREA
         LA    R5,IFRBUFR             POINT TO THE TEMPLET
         LA    R2,TEXT-IFRBUFR        LENGTH OF TEMPLET
         BCTR  R2,0                    -1 FOR THE MOVE
         EX    R2,MOVE                MOVE TEMPLET TO OUTPUT BUFFER
         L     R14,R14SAVE            RESTORE THE RETURN REGISTER
         BR    R14                    THE DSCP MSG BUFFER IS SET UP
         DROP  R11
*
* *******************************************************************
*
FREMERR  EQU *
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP002),DCJSP002
         LA    R2,L'DCJSP002          FOR CORRECT HDRMLENG
         B     OPERMSG
*
GETERR   EQU *
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP012),DCJSP012
         LA    R2,L'DCJSP012          FOR CORRECT HDRMLENG
         B     OPERMSG
*
CMDERR   EQU   *          THERE ARE NOT ENOUGH ENTRIES IN THE PDB
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP010),DCJSP010
         LA    R2,L'DCJSP010          FOR CORRECT HDRMLENG
         B     OPERMSG
*
CMDSPERR EQU   *          THE FORMAT OF 'SP = ...,' WAS NOT CORRECT
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP011),DCJSP011
         LA    R2,L'DCJSP011          FOR CORRECT HDRMLENG
         B     OPERMSG
*
CMDAPERR EQU   *          THE FORMAT OF 'AP = ...,' WAS NOT CORRECT
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP004),DCJSP004
         LA    R2,L'DCJSP004          FOR CORRECT HDRMLENG
         B     OPERMSG
*
NOCMD    EQU   *          THERE IS NO COMMAND
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP005),DCJSP005
         LA    R2,L'DCJSP005          FOR CORRECT HDRMLENG
         B     OPERMSG
*
MQSERR   EQU   *          THE RETURN CODE FROM DSIMQS WAS NOT ZERO
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP006),DCJSP006
         LA    R2,L'DCJSP006          FOR CORRECT HDRMLENG
         IC    R15,MQSERRC            GET DSIMQS ERROR RC
         STC   R15,COMMAND+33         STORE RC IN MSG
         SRL   R15,4
         STC   R15,COMMAND+32         STORE RC IN MSG
         NI    COMMAND+32,X'0F'
         NI    COMMAND+33,X'0F'
         TR    COMMAND+32(2),TRANSTBL
         B     OPERMSG
*
SPNAMER  EQU   *          THE SPNAME IS 0 OR > 8
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP007),DCJSP007
         LA    R2,L'DCJSP007          FOR CORRECT HDRMLENG
         B     OPERMSG
*
APNAMER  EQU   *          THE APPLNAME IS 0 OR > 8
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP008),DCJSP008
         LA    R2,L'DCJSP008          FOR CORRECT HDRMLENG
         B     OPERMSG
*
NOTOPER  EQU   *          THE INPUT IS NOT FROM A TERMINAL
         MVI   ERRINDC,X'01'          ERROR ENCOUNTERED
         MVC   COMMAND(L'DCJSP009),DCJSP009
         LA    R2,L'DCJSP009          FOR CORRECT HDRMLENG
         B     OPERMSG
* *****************************************************************
* SEND A MESSAGE TO THE TERMINAL IMMED AREA
* *****************************************************************
*
OPERMSG  EQU   *
         CLI   ERRINDC,X'01'           WAS ERROR ENCOUNTERED ?
         BNE   PUTGOOD                 NO - PSCP FINISHED OK
         B     CONTIMMD
PUTGOOD  MVC   COMMAND(L'DCJSP001),DCJSP001 MSG OF GOOD ENDING PSCP
         LA    R2,L'DCJSP001                   FOR CORRECT HDRMLENG
CONTIMMD LA    R4,BUFFER               R4 POINTS TO OUT BUF
         USING BUFHDR,R4
         STH   R2,HDRMLENG             PUT LENGTH IN HDRMLENG
*
         DSIPSS SWB=CWBSWB,BFR=(R4),TYPE=OUTPUT
*
*******************************************************************
*
RESTOR   EQU   *
         SR    R15,R15                ZERO R15
         L     R13,4(R13)
         ST    R15,16(R13)            SET RC TO ZERO
         LM    R14,R12,12(R13)
         BR    R14
* *********************************************************************
* *********************************************************************
*
```

```
SENDSCP  EQU   *
         ST    R14,R14SAVE             SAVE THE RETURN REGISTER
* NOW SEND THE MESSAGE TO THE DSCP
*
         L     R7,TIBTVB              ADDR OF TVB
         USING DSITVB,R7              BASE FOR TVB
         L     R4,TVBMVT              GET MY MVT ADDR
         USING DSIMVT,R4              BASE FOR MVT
*
SENDDST  DSIMQS SWB=CWBSWB,BFR=(R11),TASKID=TARGDST
         DROP  R4
         DROP  R7
         L     R14,R14SAVE            RESTORE THE RETURN REGISTER
         BR    R14                     THE MESSAGE IS ON ITS WAY
* ****************************************************************
*
*
GETDBUF  EQU   *
         ST    R14,R14SAVE             SAVE THE RETURN REGISTER
* *****************************************************************
* EVERYTHING SEEMS TO CHECK OUT OK
* NOW GET A BUFFER FOR THE IFR
* ****************************************************************
         L     R7,TIBTVB              ADDR OF TVB
         USING DSITVB,R7              BASE FOR TVB
         L     R4,TVBMVT              GET MY MVT ADDR
         USING DSIMVT,R4              BASE FOR MVT
*
* GET STORAGE FOR IFR
         DSIGET LV=(R6),A=GOTNBADR,Q=NO,SP=0
         LTR   R15,R15
         BNZ   GETERR
         DROP  R4
         DROP  R7
         L     R14,R14SAVE            RESTORE THE RETURN REGISTER
         BR    R14                    THE DSCP MSG BUFFER IS SET UP
*
* ********************************************************************
FREDBUF  EQU   *
         ST    R14,R14SAVE             SAVE THE RETURN REGISTER
         SR    R6,R6
         LH    R6,IFRBLENG            LENGTH OF BUFFER
         L     R7,TIBTVB              ADDR OF TVB
         USING DSITVB,R7              BASE FOR TVB
         L     R4,TVBMVT              GET MY MVT ADDR
         USING DSIMVT,R4              BASE FOR MVT
*
* FREE THE IFR BUFFER
         DSIFRE LV=(R6),A=FREMADDR,Q=NO,SP=0
         LTR   R15,R15
         BNZ   FREMERR
         DROP  R4
         DROP  R7
         L     R14,R14SAVE            RESTORE THE RETURN REGISTER
         BR    R14                    THE BUFFER HAS BEEN FREED
* ****************************************************************
* ****************************************************************
*
MOVE     MVC   0(,R4),0(R5)      FROM R5 TO R4
```

```
*
* *************************************************************
* DECLARES
         LTORG
*TARGDST  DC CL8'NVPCTASK'           DST TASK ID TO EXECUTE COMMAND
TARGDST  DC CL8'DSIGDS'             DST TASK ID TO EXECUTE COMMAND
IFRBUFR  EQU  *
INTRNLRQ DC Y(IFRCODCR)              IFR CODE FOR CROSS TASK QUEUE
*TARGDSCP DC CL9'NVPCDSPC'          MODULE NAME TO EXECUTE CMD
TARGDSCP DC CL9'DSISPCFD'            MODULE NAME TO EXECUTE CMD
SPNLNGTH DC   XL1'09'
SPNAME   DC   CL8'        '
APNLNGTH DC   XL1'09'
APNAME   DC   CL8'        '
TXTLNGTH DC   XL1'240'
TEXT     DC   CL240' '
TXTEND   EQU *              ** END OF MESSAGE TO DSCP **
TRANSTBL DC  C'0123456789ABCDEF'
SP       DC CL2'SP'
AP       DC CL4'APPL'
* MESSAGES
***************************************************************
*
DCJSP001 DC    C'PSCP FINISHED SUCCESSFULLY'
DCJSP002 DC    C'PSCP CANNOT OBTAIN STORAGE - EXECUTION STOPPED '
DCJSP003 DC    C'UNKNOWN SOURCE INVOKED PSCP - EXECUTION STOPPED '
DCJSP004 DC    C'FORMAT OF AP = ..., IS NOT CORRECT'
DCJSP005 DC    C'TEXT MESSAGE MUST BE 1 TO 80 CHARACTERS'
DCJSP006 DC    C'EXECUTION STOPPED - DSIMQS RC =''  '''
DCJSP007 DC    C'SP NAME MUST BE 1 TO 8 CHARACTERS'
DCJSP008 DC    C'APPL NAME MUST BE 1 TO 8 CHARACTERS'
DCJSP009 DC    C'NVPC INVOKED FROM AN UNKNOWN SOURCE'
DCJSP010 DC    C'CMD FORMAT IS: NVPC SP=(SPNAME),APPL=(APPLNAME),TEXT'
DCJSP011 DC    C'FORMAT OF SP = ..., IS NOT CORRECT'
DCJSP012 DC    C'RETURN CODE FROM DSIFREE NOT ZERO'
AUTOWORK DSECT                       POINTED TO BY CWBADATD
BUFFER   EQU   *
         ORG   *+(BUFHDRND-BUFHDR)
COMMAND  EQU   *
CMDMSG   DS    CL80' '
BUFEND   EQU   *
SPNPTR   DS    F                     POINTER TO SP NAME
APNPTR   DS    F                     POINTER TO APPL NAME
TXTPTR   DS    F                     POINTER TO THE MSG TEXT
TXTLENG  DS    XL1                   LENGTH OF MSG TEXT
TXTDISP  DS    XL1                   DISPLACEMENT TO THE TEXT
R14SAVE  DS    F                     R14 SAVE AREA
GOTNBADR DS    F                     ADDRESS RETURNED FROM DSIGET
BLDIFRAD DS    F                     ADDRESS TO BUILD IFR BUFHDR
FREMADDR DS    F                     ADDR OF DSIFRE TO FREE
RETCODE  DS    F                     RETURN CODE
IFRBLENG DS    XL2                   CALCULATED LGTH OF IFR BUFFER
GETBLENG DS    XL2                   LENGTH OF BUFFER FOR DSIGET
SPNLENG  DS    XL1                   LENGTH OF SP NAME
APNLENG  DS    XL1                   LENGTH OF APPL NAME
ERRINDC  DS    CL1                   ERROR INDICATOR
MQSERRC  DS    XL1                   DSIMQS ERROR RC SAVE AREA
AUTOEND  EQU   *
         END
```

NetView Sample Data Services Command Processor (DSCP)

```
         TITLE  'COPYRIGHT INTERNATIONAL BUSINESS MACHINES CORPORATION'
* ****************************************************************
*        API Sample Program - (C) Copyright IBM Corp. 1986, 1987
*        SAMPLE PROGRAM - NO WARRANTY EXPRESSED OR IMPLIED
* You are hereby licensed to use, reproduce, and distribute these sample
* programs as your needs require.    IBM does not warrant the suitability
* or integrity of these sample programs and accepts no responsibility for
* their use for your applications.   If you choose to copy and redistribute
* significant portions of these sample programs, you should preface such
* copies with this copyright notice.
* ****************************************************************
*
         PRINT NOGEN
 DSICBS  DSICBH,DSICWB,DSIDSRB,DSIDSB,DSIPDB,DSISWB,DSITIB,DSITVB
 DSICBS  DSIMVT,DSISVL,DSIIFR
DSISPCFD CSECT
         USING *,R15
         B     SAVEREGS
         DC    CL10'DSISPCFD'
         DC    CL42'COPYRIGHT INTERNATIONAL BUSINESS MACHINES'
         DC    CL18'CORPORATION, 1986,1987'
         DC    C'&SYSDATE'
*
* ****************************************************************
*
* THIS DATA SERVICE COMMAND PROCESSOR IS DIVIDED INTO TWO SECTIONS.
* NORMAL INPUT INTO THE FIRST SECTION IS AN IFR.
* THE IFR CONTAINS A SERVICE POINT NAME, AN APPLICATION NAME AND
* MESSAGE TEXT.
*
* OUTPUT OF THE FIRST SECTION IS A NETWORK MANAGEMENT VECTOR TRANSPORT
* (NMVT) THAT IS SENT TO THE SERVICE POINT AND APPLICATION NAMED IN
* THE INPUT IFR.
* WHEN THE NMVT IS SENT, THE OPERATOR IS NOTIFIED.
*
* NORMAL INPUT INTO THE SECOND SECTION IS A REPLY NMVT.
* THE NMVT CONTAINS ONE OR MORE REPLY MESSAGES FROM THE APPLICATION
* TO WHICH THE MESSAGE TEXT WAS SENT BY THE FIRST SECTION.
*
* NORMAL OUTPUT OF THE SECOND SECTION IS A DISPLAY OF THE REPLY
* MESSAGES RECEIVED FROM THE APPLICATION.
* SEVERAL MESSAGES ARE DISPLAYED TO THE OPERATOR ON UNEXPECTED
* CONDITIONS.
*
* ****************************************************************
         PRINT GEN
R15      EQU   15
R14      EQU   14
R13      EQU   13
R12      EQU   12
R11      EQU   11
R10      EQU   10
R9       EQU    9
R8       EQU    8
R7       EQU    7
R6       EQU    6
```

```
R5       EQU   5
R4       EQU   4
R3       EQU   3
R2       EQU   2
R1       EQU   1
R0       EQU   0
*
SAVEREGS EQU   *
         STM   R14,R12,12(R13)
         DROP  R15
         LR    R12,R15
         USING DSISPCFD,R12              MODULE ADDRESSABILITY
         LR    R3,R1                     BASE FOR CWB
         USING DSICWB,R3
         LA    R2,CWBSAVEA               GET MY SAVEAREA ADDR
         ST    R13,4(R2)                 BACKWARD CHAIN
         ST    R2,8(R13)                 FORWARD POINTER
         LR    R13,R2                    THIS SAVEA IN R13
         XC    8(4,R13),8(R13)           ZERO FORWARD POINTER
         L     R4,CWBTIB                 GET TIB ADDR
         USING DSITIB,R4
         L     R6,TIBTVB                 GET TVB ADDR
         USING DSITVB,R6
*
* IF THIS IS THE INITIAL INVOCATION,GETMAIN AN AREA FOR THE CNMI
* BUFFER AND VERIFY THE OPERANDS. IF EVERYTHING IS OK,
* ISSUE DSIZCSMS TO SEND THE REQUEST.
*
         L     R5,CWBDSRB                GET MY DSRB ADDR
         USING DSIDSRB,R5                BASE THE DSRB
         L     R7,TVBMVT                 GET MY MVT ADDR
         USING DSIMVT,R7                 BASE FOR MVT
         L     R11,DSRBUSER              CHECK FOR GOTTEN STORAGE
         CLI   DSRBFNCD,DSRBFNRM         INITIAL INVOCATION ?
         BNE   CNMIRPLY                  NO, GO HANDLE REPLY
         LTR   R11,R11                   ALREADY GOTTEN?
         BNZ   USEBUFR                   THEN USE IT
*
* GET A  WORK BUFFER
         SR    R9,R9                     ZERO R9
         LA    R9,WORKBUF                PUT LENGTH IN R9
         BAL   R14,GETMAIN               GET WORK BUFFER              *
         LTR   R15,R15                   TEST RETURN CODE
         BNZ   GETMER01                  NOT ZERO GETMAIN ERROR
         L     R11,DSRBUSER              GET POINTER TO WORK BUFFER
         USING WORKING,R11               BASE
*
         LA    R8,MSGBUFH                POINT TO OPER BUFFER TO SETUP
         BAL   R14,SETUP1                SET UP THE MSG OP MSG BUFFER *
*        BAL   R14,OK256                 TELL OPER WORK BFR GOTTEN    *
         BAL   R14,GETNMVTB              GET NMVT IN / OUT BUFFER     *
*        BAL   R14,OK1024                TELL OPER NMVT BFR GOTTEN    *
USEBUFR  BAL   R14,SETUP2                SET UP THE OUTNMVT BUFHDR    *
         BAL   R14,BUILDRU               SET UP THE OUTNMVT TEMPLET   *
         BAL   R14,FINDNAMS              MOVE NAMES TO NMVT AND       *
*                                        MOVE THE CMD TEXT TO NMVT    *
*        BAL   R14,NMVTBILT              TELL OPER NMVT READY TO SEND *
         BAL   R14,SENDRU                SEND THE NMVT TO THE SP      *
         LTR   R15,R15                   CK RETURN CODE
         BZ    SENDOK                 NO, SKIP PRINTING RETURN CODES
         BAL   R14,ZCRCMSG            TELL OPER ZCSMS RETURN CODE   *
         B     RESTEXIT               AND THEN GET OUT
SENDOK   EQU   *
         BAL   R14,ZCOKMSG            TELL OPER CMD SENT
*
* RU SCHEDULED OK, MOVE OK MESS TO OUT BUF
         LA    R9,STARTDT             WORKAREA DATETIME SAVE
         DSIDATIM AREA=(R9),FORMAT=EBCDIC
*
         B     RESTEXIT               RESTORE REGS AND EXIT
*
* ***************************************************************
* REGISTERS CURRENTLY SET UP
*                USING     R3 IS CWB ADDRESS
*                USING     R4 IS TIB ADDRESS
*                USING     R5 IS DSRB ADDRESS
*                USING     R6 IS TVB ADDRESS
*                USING     R7 IS MVT ADDRESS
*                USING    R11 IS WORKING BUFFER ADDRESS
*                USING    R12 IS MODULE BASE
*
* ***************************************************************
*  COPY THE HEADER FROM THE COMMAND BUFFER TO THE OPERATOR MSG BUFFER
*
SETUP1   ST    R14,R14SAVE             SAVE THE BAL REGISTER
         L     R10,CWBBUF              POINT TO WHERE IT IS
         LA    R9,BUFHDRND-BUFHDR      HOW LONG IT IS
         BCTR  R9,0                    -1 FOR THE MOVE
         EX    R9,MOVE                 MOVE THE HEADER
*
* INITIALIZE THE OUTPUT BUFFER HEADER FOR MESSAGES TO THE OST
*
         USING BUFHDR,R8
         LA    R10,BUFHDRND-BUFHDR     LENGTH OF BUFFER HEADER
         STH   R10,HDRTDISP            STORE OFFSET TO MSG IN HEADER
         LA    R10,256                 LENGTH OF BUFFER
         STH   R10,HDRBLENG            BUF LENGTH IN HEADER
         MVI   HDRMTYPE,HDRTYPEU      INIT MESSAGE TYPE TO USER
         MVI   HDRIND,X'00'           ZERO INDICATORS IN HEADER
         MVC   HDRDOMID(8),MVTCURAN   DOMAIN ID IN HEADER
         XC    HDRPOI(L'HDRPOI),HDRPOI ZERO POI INFO IN HEADER
         XC    HDRTSTMP(4),HDRTSTMP   PUT A PACKED ZERO
         MVI   HDRTSTMP+3,X'0C'       INTO THE TIME STAMP
         DROP  R8
         L     R14,R14SAVE            RESTORE THE RETURN ADDR
         BR    R14                    GO BACK IN LINE
*
*        END   SETUP1
*
* ***************************************************************
* PUT OUT OKMSG003
*
NMVTBILT ST    R14,R14SAVE            SAVE THE BAL REGISTER
         LA    R8,MSGBUFH             POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'OKMSG005          GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG            IN OUTBUF
         MVC   BUFHDRND(L'OKMSG005),OKMSG005   MSG IN OUT BUF
         BAL   R14,TYPEMSG              PRINT MSG IN R8 TO OPERATOR    *
         L     R14,R14SAVE              RESTORE THE RETURN ADDR
         BR    R14                      GO BACK IN LINE
*
*        END   NMVTBILT
*
* ***************************************************************
* PUT OUT OKMSG003
*
OK256    ST    R14,R14SAVE              SAVE THE BAL REGISTER
         LA    R8,MSGBUFH               POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'OKMSG003            GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG              IN OUTBUF
         MVC   BUFHDRND(L'OKMSG003),OKMSG003   MSG IN OUT BUF
         BAL   R14,TYPEMSG              PRINT MSG IN R8 TO OPERATOR    *
         L     R14,R14SAVE              RESTORE THE RETURN ADDR
         BR    R14                      GO BACK IN LINE
*
*        END   OK256
*
* ***************************************************************
* TELL OPERATOR DSIZCSMS MAJOR AND MINOR RETURN CODE
*
ZCRCMSG  ST    R14,R14SAVE              SAVE THE BAL REGISTER
         LA    R8,MSGBUFH               POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'ZCRC001             GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG              IN OUTBUF
         MVC   BUFHDRND(L'ZCRC001),ZCRC001   MSG IN OUT BUF
         STC   R15,BUFHDRND+17          STORE MAJOR RC IN MSG
         SRL   R15,4
         STC   R15,BUFHDRND+16          STORE MAJOR RC IN MSG
         NI    BUFHDRND+16,X'0F'
         NI    BUFHDRND+17,X'0F'
         TR    BUFHDRND+16(2),TRANSTBL
         STC   R0,BUFHDRND+22           STORE MINOR RC IN MSG
         SRL   R0,4
         STC   R0,BUFHDRND+21           STORE MINOR RC IN MSG
         NI    BUFHDRND+21,X'0F'
         NI    BUFHDRND+22,X'0F'
         TR    BUFHDRND+21(2),TRANSTBL
         BAL   R14,TYPEMSG              PRINT MSG IN R8 TO OPERATOR    *
         L     R14,R14SAVE              RESTORE THE RETURN ADDR
         BR    R14                      GO BACK IN LINE
*
*        END   ZCRCMSG
*
* ***************************************************************
* TELL OPERATOR DSIMQS MAJOR AND MINOR RETURN CODE
*
MQSRCMSG ST    R14,R14SAVE              SAVE THE BAL REGISTER
         LA    R8,MSGBUFH               POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'MQSRC001            GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG              IN OUTBUF
         MVC   BUFHDRND(L'MQSRC001),MQSRC001   MSG IN OUT BUF
         STC   R15,BUFHDRND+17          STORE MAJOR RC IN MSG
         SRL   R15,4
         STC   R15,BUFHDRND+16         STORE MAJOR RC IN MSG
         NI    BUFHDRND+16,X'0F'
         NI    BUFHDRND+17,X'0F'
         TR    BUFHDRND+16(2),TRANSTBL
         BAL   R14,TYPEMSG             PRINT MSG IN R8 TO OPERATOR    *
         B     RESTEXIT                TIME TO LEAVE FOR GOOD
*
*        END   MQSRCMSG
*
* *************************************************************
* PUT OUT OKMSG003
*
OK1024   ST    R14,R14SAVE             SAVE THE BAL REGISTER
         LA    R8,MSGBUFH              POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'OKMSG004           GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG             IN OUTBUF
         MVC   BUFHDRND(L'OKMSG004),OKMSG004   MSG IN OUT BUF
         BAL   R14,TYPEMSG             PRINT MSG IN R8 TO OPERATOR    *
         L     R14,R14SAVE             RESTORE THE RETURN ADDR
         BR    R14                     GO BACK IN LINE
*
*        END   OK1024
*
* *************************************************************
* FIRST TIME IN,  BETTER GET SOME STORAGE FOR THE  NMVT BUFFER
*
GETNMVTB ST    R14,R14SAVE             SAVE THE BAL REGISTER
         SR    R9,R9                   ZERO R9
         LA    R9,1024                 PUT LENGTH IN R9
         BAL   R14,GETMAIN             GET 1024 BYTE BUFFER           *
         LTR   R15,R15                 TEST RETURN CODE
         BNZ   GETMER02                NOT ZERO GETMAIN ERROR
*  COPY THE HEADER FROM THE COMMAND BUFFER TO THE NMVT BUFFER
         L     R8,DSRBUSER             GET ADDR OF GOTTEN BUFFER
         ST    R11,DSRBUSER            RESTORE ADDR OF WORK BFR
         ST    R8,RUBADDR              STORE BUFFER ADDRESS
         USING RUBUFFER,R8
         LA    R10,REPLYRU             FIND REPLY BUFFER ADDRESS
         ST    R10,ZINPUT              SAVE IT FOR THE DSIZCSMS
         DROP  R8
         USING BUFHDR,R8
         L     R10,CWBBUF              POINT TO WHERE IT IS
         SR    R9,R9
         LA    R9,BUFHDRND-BUFHDR      HOW LONG IT IS
         BCTR  R9,0                    -1 FOR THE MOVE
         EX    R9,MOVE                 MOVE THE HEADER
         SR    R9,R9
         LA    R9,BUFHDRND-BUFHDR      HOW LONG IT IS
         STH   R9,HDRTDISP             OFFSET TO TEXT
         DROP  R8
         DROP  R6
         L     R14,R14SAVE             RESTORE THE RETURN ADDR
         BR    R14                     GO BACK IN LINE
*
*        END   GETNMVTB
*
* *************************************************************
*
```

```
SETUP2   ST    R14,R14SAVE            SAVE THE BAL REGISTER
         L     R2,CWBBUF             ADDR OF INPUT COMMAND BUFFER
         USING BUFHDR,R2
         AH    R2,HDRTDISP           GET DISPLACEMENT TO TEXT
         DROP  R2
         USING INBUFFER,R2
         L     R6,RUBADDR             POINT TO RU BUFFER AREA
         USING BUFHDR,R6
*  CALCULATE THE LENGTH OF THE BUFFER TO BE SENT TO THE SP
         L     R8,RULENGTH            LENGTH OF THE INPUT BUFFER
         ST    R8,ZLENGTH             FOR ZCSMS
         SR    R9,R9
         IC    R9,TARGCMDL            LENGTH OF COMMAND TEXT
         BCTR  R9,0                   -1 FOR THE LENGTH
         AH    R9,CMDNMVTL            CALCULATE THE RU LENGTH
         STH   R9,HDRMLENG            MSG LENGTH IN THE BUFFER
         LA    R9,(BUFHDRND-BUFHDR)(R9) CALCULATED LENGTH OF BUFFER
         STH   R9,HDRBLENG            BUF LENGTH IN THE BUFFER
         ST    R9,WKBFLGTH            BUF LENGTH IN THE BUFFER
         DROP  R6
         L     R14,R14SAVE            RESTORE THE RETURN ADDR
         BR    R14                    GO BACK IN LINE
*
*        END   SETUP2
*
* ***************************************************************
*  BUILD THE FOREWARD RU
*
BUILDRU  ST    R14,R14SAVE            SAVE THE BAL REGISTER
         L     R6,RUBADDR             POINT TO RU BUFFER AREA
         USING RUBUFFER,R6
         LA    R10,NMVT               POINT TO THE TEMPLET NMVT
         USING NMVT,R10
         SR    R9,R9
         LH    R9,CMDNMVTL            GET LENGTH TO MOVE
         BCTR  R9,0                   LESS ONE
         LA    R8,RUOUT               ADDR OF START OF NMVT RU
         ST    R8,ZRU                 STORE ADDRESS FOR ZCSMS
* MOVE R9 CHARACTERS FROM WHERE R10 POINTS TO WHERE R8 POINTS
         EX    R9,MOVE                MOVE THE NMVT UP TO SV 31
         DROP  R6
         DROP  R10
         L     R14,R14SAVE            RESTORE THE RETURN ADDR
         BR    R14                    GO BACK IN LINE
*
*        END   BUILDRU
*
* ****************************************
* ***************************************************************
*  FIND AND MOVE THE SP AND APPL NAMES INTO THE RU
* AND  FIND AND MOVE THE COMMAND TEXT INTO THE RU
*
FINDNAMS ST    R14,R14SAVE            SAVE THE BAL REGISTER
         SR    R9,R9
         LA    R9,7(0,0)              IT IS ALWAYS 8 LONG
         LA    R10,TARGSPN            POINT TO WHERE THE NAME IS
         LA    R8,SPDEST              ADDR OF SP NAME
         ST    R8,ZDEST               ADDR OF SP NAME FOR ZCSMS
         ST    R8,ZTARGET             POINT TO DSIZCSMS TARGET FIELD
```

```
* MOVE 8 CHARACTERS FROM WHERE R10 POINTS TO WHERE R8 POINTS
         EX    R9,MOVE                 MOVE SP DEST NAME
*  SAVE THE TARGET APPLICATION/MANAGER NAME
FINDAP   SR    R9,R9
         LA    R9,7(0,0)               IT IS ALWAYS 8 LONG
         LA    R10,TARGAPN             POINT TO WHERE THE NAME IS
*
*    R6 IS BASED ON THE RUBUFFER DSECT AND POINTS TO THE OUTPUT BUFFER
*    R2 IS BASED ON THE INBUFFER DSECT AND POINTS TO THE INPUT DATA
*
*   PUT THE TARGET APPL/MGR NAME IN THE 50 SV
*       R10 ALREADY POINTS TO THE APPL NAME
         L     R6,RUBADDR              POINT TO RU BUFFER AREA
         USING NMVT,R6
         LA    R8,SV50DATA             POINT TO THE SV 50 DATA FIELD
         LA    R9,7(0,0)               IT IS ALWAYS 8 LONG
* MOVE R9 CHARACTERS FROM WHERE R10 POINTS TO WHERE R8 POINTS
         EX    R9,MOVE                 MOVE TARGETNAME
*   PUT THE TARGET APPL/MGR NAME IN THE ZCSMS TARGET AREA
*       R10 ALREADY POINTS TO THE APPL NAME
         LA    R8,APPLDEST             POINT TO TARGET NAME
         LA    R9,7(0,0)               IT IS ALWAYS 8 LONG
* MOVE R9 CHARACTERS FROM WHERE R10 POINTS TO WHERE R8 POINTS
         EX    R9,MOVE                 MOVE TARGETNAME
*
* ***************************************
* ************************************************************
*   BUILD THE COMMAND MESSAGE SUBVECTOR
         SR    R9,R9
         IC    R9,TARGCMDL             LENGTH OF PARAMETER
         BCTR  R9,0                    -1 FOR THE LENGTH
         AH    R9,CMDNMVTL             CALCULATE THE NEW LENGTH
         ST    R9,ZRULENG              STORE IT IN THE BUFFER
         SR    R9,R9
         IC    R9,TARGCMDL             LENGTH OF PARAMETER
         AH    R9,LL8061               CALCULATE THE NEW LENGTH
         STH   R9,LL8061               STORE IT IN THE MV-LL
         LA    R8,SV31DATA             POINT TO SV31 LOCATION
         LA    R10,TARGCMD             ADDRESS OF START OF CMD TEXT
         SR    R9,R9                   ZERO R9
         IC    R9,TARGCMDL             LENGTH OF PARAMETER
         BCTR  R9,0                    -1 FOR THE LENGTH
         LA    R9,2(,R9)               ADD 2 TO LENGTH
         STC   R9,LSV31                STORE THE LENGTH OF THE SV
         SR    R9,R9                   ZERO R9
         IC    R9,TARGCMDL             LENGTH OF PARAMETER
         BCTR  R9,0                    -1 FOR THE MOVE
* MOVE R9 CHARACTERS FROM WHERE R10 POINTS TO WHERE R8 POINTS
         EX    R9,MOVE                 MOVE PARAMETER
         L     R14,R14SAVE             RESTORE THE RETURN ADDR
         DROP  R6
         BR    R14                     GO BACK IN LINE
*
*        END   FINDNAMS
*
* ************************************************************
*   SEND THE NMVT
*
SENDRU   ST    R14,R14SAVE             SAVE THE BAL REGISTER
```

```
         L     R6,RUBADDR             POINT TO RU BUFFER AREA
         USING RUBUFFER,R6
*  ISSUE   DSIZCSMS TO SEND FORWARD RU
ISSUECNM DSIZCSMS  SWB=CWBSWB,DSRB=(R5),INPUT=ZINPUT,                  *
               LENGTH=ZLENGTH,RU=ZRU,RULENG=ZRULENG,                   *
               DEST=ZDEST,TARGET=ZTARGET
         L     R14,R14SAVE            RESTORE THE RETURN ADDR
         BR    R14                    GO BACK IN LINE
         DROP  R6
*
*        END   SENDRU
*
* ****************************************************************
*  DSIZSMS EXECUTED OK
*  PREPARE MESSAGE
*
ZCOKMSG  ST    R14,R14SAVE            SAVE THE BAL REGISTER
         LA    R8,MSGBUFH             POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'OKMSG001          GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG            IN OUTBUF
         MVC   BUFHDRND(L'OKMSG001),OKMSG001   MSG IN OUT BUF
         BAL   R14,TYPEMSG            PRINT MSG IN R8 TO OPERATOR    *
         DROP  R2
         DROP  R8
         L     R14,R14SAVE            RESTORE THE RETURN ADDR
         BR    R14                    GO BACK IN LINE
*
*        END   ZCOKMSG
*
* ****************************************************************
*
         EJECT
*
*
         DROP  R4
*
CNMIRPLY EQU   *
*
* REGISTERS CURRENTLY SET UP
*                  USING     R3 IS CWB ADDRESS
*                  USING     R5 IS DSRB ADDRESS
*                  USING     R7 IS MVT ADDRESS
*                  USING    R11 IS WORKING BUFFER ADDRESS
*                  USING    R12 IS MODULE BASE
*
*
*
*  IF THE REQUESTED  FUNCTION WAS SUCCESSFULLY COMPLETED, BUILD THE
*  APPROPRIATE  COMMAND TO NOTIFY THE OPERATOR
*
         BAL   R14,CKINPUT            CHECK FOR VALID CNMI INPUT     *
         BAL   R14,CHKDATA            CHECK THE RECEIVED DATA        *
         BAL   R14,BLDRMSG            BUILD THE OPERATOR MSG         *
         LA    R8,MSGBUFH             POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         BAL   R14,TYPEMSG            PRINT MSG IN R8 TO OPERATOR    *
         DROP  R8
         BAL   R14,MULTIPLE           CK FOR AND PRINT ADDITIONAL    *
```

```
*                                        MESSAGES IF ANY
         B     RESTEXIT                  LEAVE FOR GOOD
*
*        END   CNMIRPLY
*
* ************************************************************
*     IF UNSOLICITED THEN DISPLAY ERROR MESSAGE
*
CHKDATA  ST    R14,R14SAVE               SAVE THE BAL REGISTER
         L     R10,DSRBINPT               GET ADDR OF CNMI BUF
         USING BUFHDR,R10
         AH    R10,HDRTDISP               FIND START OF TEXT
         DROP  R10
         USING DELIVRRU,R10
         LA    R2,LLMV0061                POINT TO THE MV
         ST    R2,ADDR0061                SAVE POINTER TO 0061 MV
         DROP  R10
         USING LLMV0061,R2                BASE FOR MV
*  IF IT IS NOT A NETVIEW/PC (X'0061') REPLY
*    THEN DISPLAY ERROR MESSAGE
         CLC   IDMV0061,NVPCREPL          NVPC REPLY?
         BNE   NOTREPLY                   NO, TELL OPERATOR
*
* ---------> IF IT IS NOT X'1300' THEN DISPLAY ERROR MESSAGE
RPLYKEY  LA    R8,LENG0061               POINT TO MV 0061 LEN WORK AREA
         LA    R10,LLMV0061              POINT TO THE 0061 ADDRESS
         ST    R10,ADDR0061              SAVE 0061 ADDRESS
         DROP  R2
         SR    R9,R9
         LA    R9,1                      TO MOVE 2 BYTES
         EX    R9,MOVE                   MOVE THE LENGTH TO WORK AREA
         AH    R10,LENG0061              ADD THE LENGTH OF THE MV
         ST    R10,ADDR1300              SAVE ADDR OF MV 1300
         USING LLMV1300,R10              BASE THE MESSAGE MV
         CLC   IDMV1300,UNFORMMV         UNFORMATTED REPLY?
         BE    MV1309                    YES, TELL OPERATOR
         CLC   IDMV1300,FORMREPL         FORMATTED REPLY?
         DROP  R10
         BNE   NOTFMT                    NO, CK IF SENSE
         LA    R8,LENG1300               POINT TO MV 1300 LEN WORK AREA
         SR    R9,R9
         LA    R9,1                      TO MOVE 2 BYTES
         EX    R9,MOVE                   MOVE THE LENGTH TO WORK AREA
         L     R14,R14SAVE               RESTORE THE RETURN ADDR
         BR    R14                       GO BACK IN LINE
*
NOTFMT   EQU   *
         L     R10,ADDR0061              POINT TO THE 0061 MV
         USING LLMV,R10                  BASE THE MESSAGE MV
         LA    R10,LSV                   POINT TO NEXT SV
         DROP  R10
         USING LSV,R10                   BASE THE SV
         DROP  R2
         SR    R2,R2                     ZERO R2
         IC    R2,LSV                    GET NEXT SV LENGTH
         AR    R10,R2                    POINT R10 TO NEXT SV
         CLI   IDSV,X'7D'                IS THERE SENSE DATA
         BNE   NOTFTMSG                  NO, TELL NOT FORMATTED
         LA    R8,SVDATA                 POINT TO THE DATA
         DROP  R10
         B     SENSEMSG                TELL SENSE
*
*        END CHKDATA
*
* ****************************************************************
*
CKINPUT  ST    R14,R14SAVE              SAVE THE BAL REGISTER
         TM    DSRBFLG,DSRBTYPE         UNSOLICITED FUNCTION CODE?
         BO    UNSOL                    YES, TELL OPERATOR
         LA    R6,MSGBUFH              POINT TO OPER MSG BUFFER
         USING BUFHDR,R6
         MVI   OUTDATE,X'40'
         MVC   OUTDATE+1(OUTLEN-1),OUTDATE  BLANK BUFFER
         DSIDATIM AREA=OUTDATE,FORMAT=EBCDIC
         UNPK  OUTPRID(3),DSRBPRID(2)  UNPACK PRID
         MVC   OUTPRID+3(1),DSRBPRID+1 MOVE LAST BYTE
         OI    OUTPRID+2,X'F0'         MAKE ZONE CORRECT
         OI    OUTPRID+3,X'F0'         MAKE ZONE CORRECT
         TR    OUTPRID(4),TRANSTBL-240 MAKE ALL CHARS PRINTABLE
         UNPK  OUTRMAJ(2),DSRBRCMA+3
         MVC   OUTRMAJ+1(1),DSRBRCMA+3
         OI    OUTRMAJ,X'F0'
         OI    OUTRMAJ+1,X'F0'
         TR    OUTRMAJ(2),TRANSTBL-240
         UNPK  OUTRMIN(2),DSRBRCMI+3
         MVC   OUTRMIN+1(1),DSRBRCMI+3
         OI    OUTRMIN,X'F0'
         OI    OUTRMIN+1,X'F0'
         TR    OUTRMIN(2),TRANSTBL-240
         LA    R8,OUTLEN               LENGTH OF BUFFER
         STH   R8,HDRBLENG             TO BUF HDR
         LA    R8,OUTRU-MESSAGE        LENGTH OF MESSAGE
         STH   R8,HDRMLENG             TO BUF HDR
         LA    R8,BUFHDRND-BUFHDR      GET LENGTH OF BUFHDR
         STH   R8,HDRTDISP             DISPLACEMENT TO TEXT
         CLC   DSRBRCMI(4),RSPGOOD     GOOD MINOR RC
         BNE   AMERROR                 NO, PREPARE MESSAGES
         L     R14,R14SAVE             RESTORE THE RETURN ADDR
         BR    R14                      GO BACK IN LINE
*
AMERROR  L     R8,DSRBINPT              GET ADDR OF CNMI BUF
         AH    R8,HDRTDISP-BUFHDR(R8)   POINT 8 TO START OF RESP
SENSEMSG LA    R6,MSGBUFH               MAKE SURE R6 IS CORRECT
* ****************************************************************
* THE SENSE DATA IS DISPLAYED TO THE OPERATOR.
* THE FOLLOWING SENSE DATA IS UNIQUE TO SPCF
* 8018 0001  TARGET MANAGER NOT RECOGNIZED
* 084B 0003  THE RECEIVER IS NOT AVAILABLE
* 1003 000D  THE FUNCTION IS NOT SUPPORTED OR -
*            A CHARACTER COULD NOT BE TRANSLATED
* ****************************************************************
         UNPK  OUTRU(7),0(4,R8)         MAKE FIRST 4 BYTES READABLE
         MVC   OUTRU+7(1),3(R8)
         OI    OUTRU+6,X'F0'
         OI    OUTRU+7,X'F0'
         TR    OUTRU(8),TRANSTBL-240
         LA    R8,8                     LENGTH OF SENSE
         AH    R8,HDRMLENG              ADD TO PREFIX LENGTH
         STH   R8,HDRMLENG              IN BUFHDR
         LA    R8,MSGBUFH               POINT TO OPER MSG BUFFER
         BAL   R14,TYPEMSG              PUT OUT AN ERROR MESSAGE     *
         SR    R15,R15                  ZERO R15
         B     RESTEXIT                 TIME TO LEAVE
         DROP  R6
*
*        END   CKINPUT
*
* ***************************************************************
* PREPARE A REPLY MESSAGE FROM THE CNMI MESSAGE
*
BLDRMSG  ST    R14,R14SAVE             SAVE THE BAL REGISTER
         LA    R6,MSGBUFH              POINT TO OPER MSG BUFFER
         USING BUFHDR,R6
         L     R2,ADDR0061             POINT TO MV ID 0061
         USING LLMV0061,R2             BASE FOR MV
         CLI   IDSV,X'31'              MESSAGE SC ID?
         BNE   NOTSV31                 NO, PUT OUT ERROR MSG
         LA    R8,OUTRMAJ              START OF REPLY
         LA    R10,LSV                 ADDRESS OF FIRST 31 SV
         ST    R10,ADDRSV31            SAVE ADDR OF CURRENT SV 31
         LA    R10,SVDATA              START OF MESSAGE SV
         LA    R9,WKBUFEND-MSGBUFH     GET LENG OF MSG BUFFER
         STH   R9,HDRBLENG             STORE THE BUFFER LENGTH
         LA    R9,OUTRMAJ-MESSAGE      GET LENGTH MSG PREFIX
         STH   R9,HDRMLENG             STORE PREFIX LENGTH
         SR    R9,R9
         IC    R9,LSV                  GET THE LENGTH
         BCTR  R9,0                    MINUS 1 FOR THE LENGTH FIELD
         BCTR  R9,0                    MINUS 1 FOR THE SV ID FIELD
         AH    R9,HDRMLENG             ADD THE TEXT LENGTH
         STH   R9,HDRMLENG             STORE THE MESSAGE LENGTH
         SR    R9,R9
         IC    R9,LSV                  GET THE LENGTH
         BCTR  R9,0                    MINUS 1 FOR THE LENGTH FIELD
         BCTR  R9,0                    MINUS 1 FOR THE SV ID FIELD
         BCTR  R9,0                    MINUS 1 FOR THE MOVE
         EX    R9,MOVE                 MOVE IT
         SR    R9,R9
         L     R14,R14SAVE             RESTORE THE RETURN ADDR
         DROP  R2
         DROP  R6
         BR    R14                     GO BACK IN LINE
*
*        END   BLDRMSG
*
* ***************************************************************
*  IF THERE IS MORE THAN ONE SV, PRINT A ONE LINE MESSAGE
*  FOR EACH SV 31 AFTER THE FIRST ONE.
*
MULTIPLE ST    R14,R14SAVE             SAVE THE BAL REGISTER
CHKMORE  L     R2,ADDRSV31             POINT TO SV ID 31
         USING LSV,R2
         SR    R9,R9
         IC    R9,LSV                  GET THE SV LENGTH
         ALR   R2,R9                   ADD TO THE ADDRESS
         ST    R2,ADDRSV31             POINT TO NEXT SV ID 31
         CLI   IDSV,X'31'              MESSAGE SC ID?
         BNE   NOMORESV                NO, GET OUT
         LA    R8,MSGBUFH              ADDR OF MESSAGE OUTPUT BUFHDR
         USING BUFHDR,R8
         LA    R9,OUTRMAJ-MESSAGE      MESSAGE PREFIX LENGTH
         STH   R9,HDRMLENG             TO BUFHDR MESSAGE LENGTH
         SR    R9,R9
         IC    R9,LSV                  GET THE SV LENGTH
         BCTR  R9,0                    -1 FOR SV LENGTH FIELD
         BCTR  R9,0                    -1 FOR SV ID FIELD
         AH    R9,HDRMLENG             ADD SV 31 LENGTH AND
         STH   R9,HDRMLENG             STORE MESSAGE LENGTH
         SR    R9,R9
         IC    R9,LSV                  GET LENGTH OF SV
         BCTR  R9,0                    -1 FOR SV ID
         BCTR  R9,0                    -1 FOR SV LENGTH FIELD
         BCTR  R9,0                    -1 FOR THE MOVE
         DROP  R8
         LA    R10,SVDATA              POINT TO THE DATA
         LA    R8,OUTRMAJ              POINT TO WHERE IT GOES
         EX    R9,MOVE                 MOVE TEXT TO OUTPUT MSG BUFFER
         LA    R8,MSGBUFH              POINT TO THE BUFFER HEADER
         BAL   R14,TYPEMSG             PRINT THE MESSAGE            *
         B     CHKMORE                 CK FOR MORE SV31-S
         DROP  R2
NOMORESV L     R14,R14SAVE             RESTORE THE RETURN ADDR
         BR    R14                     GO BACK IN LINE
*
*        END   MULTIPLE
*
* *************************************************************
*     IT IS NOT A REPLY FROM NETVIEW/PC
*
NOTREPLY LA    R8,MSGBUFH              POINT TO OPER MSG BUFFER
         USING BUFHDR,R8
         LA    R9,L'NOT0061
         STH   R9,HDRMLENG
         MVC   BUFHDRND(L'NOT0061),NOT0061
         MVC   BUFHDRND+31(1),DSRBFNCD
         DROP  R8
         BAL   R14,TYPEMSG              PUT OUT AN ERROR MESSAGE    *
         SR    R15,R15                  ZERO R15
         LA    R15,8                    PUT RC IN R15
         B     RESTEXIT                LEAVE FOR GOOD
*
*        END   NOTREPLY
*
* *************************************************************
MV1309   EQU   *
* THE MAJOR VECTOR IS FOR UNFORMATTED DATA
* REGISTER 10 AND ADDR1300 POINT TO THE 1309 MV
* ADD CODE HERE TO HANDLE UNFORMATTED DATA IF REQUIRED
*
         LA    R8,MSGBUFH              POINT TO OPER MSG BUFFER
         USING BUFHDR,R8
         LA    R9,L'NOTFM002
         STH   R9,HDRMLENG
         MVC   BUFHDRND(L'NOTFM002),NOTFM002
         DROP  R8
         LA    R8,MSGBUFH               POINT TO OPER MSG BUFFER
         BAL   R14,TYPEMSG               PUT OUT AN ERROR MESSAGE     *
         SR    R15,R15                   ZERO R15
         B     RESTEXIT                  TIME TO LEAVE
*
*        END NOTFTMSG
*
* ***************************************************************
*
NOTFTMSG LA    R8,MSGBUFH               POINT TO OPER MSG BUFFER
         USING BUFHDR,R8
         LA    R9,L'NOTFM001
         STH   R9,HDRMLENG
         MVC   BUFHDRND(L'NOTFM001),NOTFM001
         DROP  R8
         LA    R8,MSGBUFH                POINT TO OPER MSG BUFFER
         BAL   R14,TYPEMSG               PUT OUT AN ERROR MESSAGE     *
         SR    R15,R15                   ZERO R15
         B     RESTEXIT                  TIME TO LEAVE
*
*        END NOTFTMSG
*
* ***************************************************************
*
UNSOL    LA    R8,MSGBUFH                POINT TO OPER MSG BUFFER
         USING BUFHDR,R8
         LA    R9,L'NOTSOL
         STH   R9,HDRMLENG
         MVC   BUFHDRND(L'NOTSOL),NOTSOL
         DROP  R8
         BAL   R14,TYPEMSG               PUT OUT AN ERROR MESSAGE     *
         SR    R15,R15                   ZERO R15
         L     R8,DSRBINPT               GET ADDR OF INPUT BUFFER
         ST    R8,FREMADDR               STORE ADDRESS TO FREE
         USING BUFHDR,R8
         LH    R9,HDRBLENG               GET LENGTH TO FREE
         DROP  R8
         BAL   R14,FREEBUF               GO BACK IN LINE             *
         B     RESTEXIT                  TIME TO LEAVE
*
*        END   UNSOL
*
* ***************************************************************
*
NOTSV31  LA    R8,MSGBUFH               POINT TO OPER MSG BUFFER
         USING BUFHDR,R8
         LA    R9,L'NOTSV031
         STH   R9,HDRMLENG
         MVC   BUFHDRND(L'NOTSV031),NOTSV031
         DROP  R8
         BAL   R14,TYPEMSG               PUT OUT AN ERROR MESSAGE     *
         SR    R15,R15                   ZERO R15
         B     RESTEXIT                  TIME TO LEAVE
*
*        END   NOTSV31
*
* ***************************************************************
*
* GET STORAGE  FOR THE LENGTH IN R9 AND RETURN THE ADDRESS IN R8
GETMAIN  LR    R10,R14                   SAVE THE BAL ADDRESS
```

```
         L     R11,DSRBUSER
         DSIGET LV=(R9),A=DSRBUSER,Q=YES,TASKA=(R6)
         LR    R14,R10                 RESTORE THE RETURN ADDR
         BR    R14                     GO BACK IN LINE
*
*        END   GETMAIN
*
* ****************************************************************
* FREE THE BUFFER ADDRESS IN REGISTER 8 FOR THE LENGTH IN REGISTER 9
FREEBUF  ST    R14,R14SAVE             SAVE THE BAL REGISTER
         L     R6,CWBTIB
         USING DSITIB,R6
         L     R6,TIBTVB                TVB ADDR
         DROP  R6
       DSIFRE LV=(R9),A=FREMADDR,TASKA=(R6)
         L     R14,R14SAVE             RESTORE THE RETURN ADDR
         BR    R14                     GO BACK IN LINE
*
*        END   FREEBUF
*
* ****************************************************************
* PRINT A MESSAGE, IN THE BUFFER POINTED TO BY R8, TO THE OPERATOR
TYPEMSG  LR    R10,R14                 SAVE THE BAL REGISTER
         L     R6,CWBTIB               GET TIB ADDR
         USING DSITIB,R6
         L     R6,TIBTVB               GET TVB ADDR
         DROP  R6
         USING DSITVB,R6
*   PUT OUT A MESSAGE
         DSIMQS SWB=CWBSWB,BFR=(R8),TASKID=DSRBOID
         DROP  R6
         LTR   R15,R15                 CK RETURN CODE
         BZ    TYPEOUT                 OK, THEN BLANK MSG AREA
         B     MQSRCMSG                TELL OPERATOR MQS RC
TYPEOUT  EQU   *
         LR    R14,R10                 RESTORE THE RETURN ADDR
         BR    R14                     GO BACK IN LINE
*
*        END   TYPEMSG
*
* ****************************************************************
*
* STANDARD EXIT
RESTEXIT EQU   *
         L     R13,4(R13)
         LM    R14,R12,12(R13)
         BR    R14
*
*        END   RESTEXIT
*
* ****************************************************************
*
FREMERR  EQU   *
* DSIFRE  FAILED
         LA    R8,MSGBUFH              POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'FMERR001           GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG             IN OUTBUF
         MVC   BUFHDRND(L'FMERR001),FMERR001   MSG IN OUT BUF
         BAL   R14,TYPEMSG             PRINT MSG IN R8 TO OPERATOR    *
         B     RESTEXIT                RESTORE REGS AND EXIT
         DROP  R8
*
*        END   FREMERR
*
* ***************************************************************
*
GETMER01 EQU   *
* DSIGET  FAILED, MUST USE THE CWB AUTOWORK AREA FOR OPER MSG
         LA    R8,CWBADATD             POINT TO OPER MESSAGE AREA
         BAL   R14,SETUP1              SET UP THE OPER MSG BUFHDR
         USING BUFHDR,R8
         LA    R9,L'GMERR002           GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG             IN OUTBUF
         MVC   BUFHDRND(L'GMERR002),GMERR002   MSG IN OUT BUF
         BAL   R14,TYPEMSG             PRINT MSG IN R8 TO OPERATOR    *
         B     RESTEXIT                RESTORE REGS AND EXIT
         DROP  R8
*
*        END   GETMER01
*
* ***************************************************************
*
GETMER02 EQU   *
* DSIGET  FAILED
         LA    R8,MSGBUFH              POINT TO THE BUFFER HEADER
         USING BUFHDR,R8
         LA    R9,L'GMERR002           GET LENGTH OF MESSAGE
         STH   R9,HDRMLENG             IN OUTBUF
         MVC   BUFHDRND(L'GMERR002),GMERR002   MSG IN OUT BUF
         BAL   R14,TYPEMSG             PRINT MSG IN R8 TO OPERATOR    *
         B     RESTEXIT                RESTORE REGS AND EXIT
*
*        END   GETMER02
*
* ***************************************************************
* ***************************************************************
* DECLARES
         LTORG
RSPGOOD  DC    A(DSRCGOOD)
RSPNGR   DC    A(DSRCNGRP)
RULENGTH DC    AL4(RUBEND-REPLYRU)     REPLY BUFFER LENGTH
*
*  REGISTER 8 MUST POINT TO THE ADDRESS THAT DATA WILL BE MOVED TO
*  REGISTER 9 MUST CONTAIN THE COUNT (-1) OF DATA TO BE MOVED
*  REGISTER 10 MUST POINT TO THE PARM THAT WILL BE MOVED
*
MOVE     MVC   0(0,R8),0(R10)             MOVE FROM R10 TO R8
*
* ***************************************************************
OKMSG001 DC    C'MESSAGE QUEUED TO THE SERVICE POINT'
OKMSG002 DC    C'INPUT COMMAND BUFFER FREED OK'
OKMSG003 DC    C'256 BYTE WORK BUFFER GOTTEN OK'
OKMSG004 DC    C'1024 BYTE NMVT BUFFER GOTTEN OK'
OKMSG005 DC    C'DSIZCSMS WILL BE ISSUED NEXT'
GMERR001 DC    C'DSIGET FOR WORK BUFFER FAILED'
GMERR002 DC    C'DSIGET FOR CNMI BUFFER FAILED'
FMERR001 DC    C'DSIFRE FOR INPUT BUFFER FAILED'
```

```
FMERR002 DC    C'DSIFRE FOR CNMI BUFFER FAILED'
ZCRC001  DC    C'DSIZCSMS RC = X''  '' ''  '''
MQSRC001 DC    C'DSIMQS RC =   X''  '''
NOT0061  DC    C'MV IS NOT 0061'
NOTSOL   DC    C'NMVT IS NOT A SOLICITED REPLY'
NOTFM001 DC    C'NMVT IS NOT RECOGNIZED'
NOTFM002 DC    C'NMVT MV ID IS 1309'
NOTSV031 DC    C'MV 1300 SV IS NOT ID 31'
ENDMSG   DC    C'NVPC DSCP ENDING'
TRANSTBL DC    C'0123456789ABCDEF'
INTRNLRQ DC    Y(IFRCODCR)            IFR CODE FORCROSS TASK CMD QUEUE
OUTFWRD  DC    CL8'FORWARD'
OUTGETM  DC    CL8'GETMAIN'
OUTFAIL  DC    CL8'FAILED'
NPCREPLY DC    XL3'41038D'             NETVIEW/PC REPLY
NVPCREPL DC    XL2'0061'
FORMREPL DC    XL2'1300'
UNFORMMV DC    XL2'1309'
*
NVPCCMD  DS    0H                       ALIGNMENT
CMDNMVTL DC    AL2(ENDSV31-NMVT)        LENGTH OF THE NMVT
NMVT     DC    XL3'41038D'              NMVT RU
RETIRED  DC    XL2'0000'
NMVTPRID DC    XL2'0000'
NMVTFLAG DC    XL1'00'
*
*                       MAJOR VECTOR 8061
LL8061   DC    AL2(END8061-LL8061)     MVID 8061 LENGTH
ID8061   DC    XL2'8061'
*
LSV06    DC    AL1(ENDSV50-LSV06)      SVID 06 LENGTH
SVID06   DC    XL1'06'
ENDSV06  EQU   *                       END OF SVID 06
*
LSV50    DC    AL1(ENDSV50-LSV50)      SVID 50 LENGTH
SVID50   DC    XL1'50'
SV50DATA DC    CL8'SAMPAPPL'           TARGET APPL/MGR NAME
ENDSV50  EQU   *                       END OF SVID 50
*
LSV31    DC    AL1(ENDSV31-LSV31)      SVID 31 LENGTH
SVID31   DC    XL1'31'
SV31DATA DS    0H                      COMMAND TEXT
ENDSV31  EQU   *                       END OF SVID 31
END8061  EQU   *                       END OF MVID 8061
*
* **************************************************************
*
INBUFFER DSECT
DSCPNAME DS    CL9                     DSCP NAME AND A BLANK
*INCMDNAM DS    CL9                    OPERATOR ENTERED CMD AND BLANK
TARGSPL  DS    XL1                     LENGTH OF TARGET SP NAME
TARGSPN  DS    CL8                     TARGET SP NAME - INIT TO BLANKS
TARGAPL  DS    XL1                     LENGTH OF TARGET APPL NAME
TARGAPN  DS    CL8                     TARGET APPL NAME-INIT TO BLANKS
TARGCMDL DS    XL1                     LENGTH OF COMMAND
TARGCMD  DS    CL240                   COMMAND TEXT UP TO 240 BYTES
*
* **************************************************************
*
```

```
RUBUFFER DSECT  1024 BYTES OF GOTTEN STORAGE POINTED TO BY RUBADDR
* HERE STARTS THE OUTPUT NMVT
RUOUT    EQU   *
*
* HERE STARTS THE INPUT RU AREA IN THE 1024 BYTE GOTTEN STORAGE
         ORG   RUOUT+400                START OF REPLY AREA
REPLYRU  EQU   *                        START OF BUFFER HEADER
         ORG   *+(BUFHDRND-BUFHDR)      END OF BUFHDR
RUIN     EQU   *
RUBEND   EQU   RUOUT+1024               END OF RU BUFFER
*
* ****************************************************************
*
         DSECT
WORKING  EQU   *   256 BYTES OF GOTTEN STORAGE POINTED TO BY DSRBUSER
RUBADDR  DS    F                        ADDRESS OF RUBUFFER
ZINPUT   DS    F                        ADDRESS OF REPLY BUFFER
ZLENGTH  DS    F                        LENGTH OF THE INPUT RU BUFFER
ZRU      DS    F                        ADRESS OF AREA WITH OUTPUT RU
ZRULENG  DS    F                        LENGTH OF IMBEDED RU BUFFER
ZDEST    DS    F                        ADRESS OF SP DEST NAME
ZTARGET  DS    F                        ADRESS OF TARGET APPL
R14SAVE  DS    F                        R14 SAVEAREA
WKBFLGTH DS    F                        WORK BUFFER LENGTH
FREMADDR DS    F                        ADDRESS FOR DSIFRE TO FREE
ADDR0061 DS    F                        ADDRESS OF THE MV ID X'0061'
LENG0061 DS    H                        LENGTH OF MV ID X'0061
ADDRSV31 DS    F                        ADDRESS OF THE SV ID X'31'
ADDR1300 DS    F                        ADDRESS OF THE MV ID X'1300'
LENG1300 DS    H                        LENG REMAINING IN MV ID X'1300
         DS    0F                       ALIGN
SPDEST   DS    CL8                      SERVICE POINT NAME
APPLDEST DS    CL8                      TARGET APPLICATION NAME
STARTDT  DS    CL17                     DATE AND TIME MESSAGE SENT
         DS    0F                       ALIGN ON A WORD
MSGBUFH  EQU   *                        START OF BUFFER HEADER
         ORG   *+(BUFHDRND-BUFHDR)      START OF TEXT
MESSAGE  EQU   *                        HERE STARTS THE OST MSG
OUTHDR   EQU   *
OUTDATE  DS    CL8  MESSAGE DATE
         DS    CL1
OUTTIME  DS    CL8  MESSAGE TIME
         DS    CL1
OUTPRID  DS    CL4  CORRELATION REQUEST ID
         DS    CL1
OUTRMAJ  DS    CL2  MAJOR RETURN CODE
         DS    CL1
OUTRMIN  DS    CL2  MINOR RETURN CODE
         DS    CL1
OUTRU    DS    CL8    SENSE BYTES IF NEGATIVE RESPONSE
         DS    CL1
OUTMSG   DS    CL80   MESSAGE TEXT GOES HERE
OUTEND   EQU   *
OUTLEN   EQU   OUTEND-OUTHDR
         ORG   OUTMSG+256
WORKBUF  EQU   *-WORKING
WKBUFEND EQU   *
*
* ****************************************************************
```

```
*
RPLYNMVT DSECT
         ORG   *+(BUFHDRND-BUFHDR)     START OF TEXT
DELIVRRU DS    XL8
NSHDR    DS    XL3
RESV     DS    XL2
PRID     DS    XL2
FLAGS    DS    XL1
*    SPCF REPLY MAJOR VECTOR AND SUBVECTOR DSECT
LLMV0061 DS    XL2                     MV  LENGTH
IDMV0061 DS    XL2                     MV  ID
         DS    XL4                     SV 44
LLMV1300 DS    XL2                     MV  LENGTH
IDMV1300 DS    XL2                     MV  ID
         ORG   LLMV1300                ANY MV-SV FOLLOWS
LLMV     DS    XL2                     MV  LENGTH
IDMV     DS    XL2                     MV  ID
LSV      DS    XL1                     SV  LENGTH
IDSV     DS    XL1                     SV  ID
SVDATA   EQU   *                       SV  DATA
*
SV7D     ORG   LLMV1300                IF NOT MV 1300 THEN MAY BE SV7D
LSV7D    DS    XL1                     LENGTH OF SV
IDSV7D   DS    XL1                     ID OF SV
* ***************************************************************
         END
```

Glossary

This glossary defines important NCP, NetView, NetView/PC, SSP, and VTAM abbreviations and terms. It includes information from the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699. Definitions from the *American National Dictionary for Information Processing* are identified by an asterisk (*). Definitions from draft proposals and working papers under development by the International Standards Organization, Technical Committee 97, Subcommittee 1 are identified by the symbol **(TC97)**. Definitions from the *CCIT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the Consultative Committee on International Telegraph and Telephone of the International Telecommunication Union, Geneva, 1980 are preceded by the symbol **(CCITT/ITU)**. Definitions from published sections of the *ISO Vocabulary of Data Processing*, developed by the International Standards Organization, Technical Committee 97, Subcommittee 1 and from published sections of the *ISO Vocabulary of Office Machines*, developed by subcommittees of ISO Technical Committee 95, are preceded by the symbol **(ISO)**.

For abbreviations, the definition usually consists only of the words represented by the letters; for complete definitions, see the entries for the words.

Reference Words Used in the Entries

The following reference words are used in this glossary:

Deprecated term for. Indicates that the term should not be used. It refers to a preferred term, which is defined.

Synonymous with. Appears in the commentary of a preferred term and identifies less desirable or less specific terms that have the same meaning.

Synonym for. Appears in the commentary of a less desirable or less specific term and identifies the preferred term that has the same meaning.

Contrast with. Refers to a term that has an opposed or substantively different meaning.

See. Refers to multiple-word terms that have the same last word.

See also. Refers to related terms that have similar (but not synonymous) meanings.

ABEND. Abnormal end of task.

abnormal end of task (ABEND). Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

ACB name. (1) The name of an ACB macroinstruction. (2) A name specified in the ACBNAME parameter of a VTAM APPL statement. Contrast with *network name*.

accept. For a VTAM application program, to establish a session with a logical unit (LU) in response to a CINIT request from a system services control point (SSCP). The session-initiation request may begin when a terminal user logs on, a VTAM application program issues a macroinstruction, or a VTAM operator issues a command. See also *acquire (1)*.

access method. A technique for moving data between main storage and input/output devices.

accounting exit routine. In VTAM, an optional installation exit routine that collects statistics about session initiation and termination.

ACF/NCP. Advanced Communications Function for the Network Control Program. Synonym for *NCP*.

acquire. (1) For a VTAM application program, to initiate and establish a session with another logical unit (LU). The acquire process begins when the application program issues a macroinstruction. See also *accept*. (2) To take over resources that were formerly controlled by an access method in another domain, or to resume control of resources that were controlled by this domain but released. Contrast with *release*. See also *resource takeover*.

active. (1) The state a resource is in when it has been activated and is operational. Contrast with *inactive, pending*, and *inoperative*. (2) Pertaining to a major or minor node that has been activated by VTAM. Most resources are activated as part of VTAM start processing or as the result of a VARY ACT command.

adapter. Hardware card that allows a device, such as a PC, to communicate with another device, such as a monitor, a printer, or other I/O device.

alert. (1) In SNA, a record sent to a system problem management focal point to communicate the existence of an alert condition. (2) In the NetView program, a high priority event that warrants immediate attention. This data base record is generated for certain event types that are defined by user-constructed filters.

alert condition. A problem or impending problem for which some or all of the process of problem determination, diagnosis, and resolution is expected to require action at a control point.

allocate. A logical unit (LU) 6.2 application program interface (API) verb used to assign a session to a conversation for the conversation's use. Contrast with *deallocate*.

API. Application program interface.

application program. (1) A program written for or by a user that applies to the user's work. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application program interface (API). (1) The formally defined programming language interface between an IBM system control program or licensed program and its user. (2) The interface through which an application program interacts with an access method. In VTAM, it is the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

ASCII. American National Standard Code for Information Interchange.

authorization exit routine. In VTAM, an optional installation exit routine that approves or disapproves requests for session initiation.

automatic logon. (1) A process by which VTAM automatically creates a session-initiation request to establish a session between two logical units (LUs). The session will be between a designated primary logical unit (PLU) and a secondary logical unit (SLU) that is neither queued for nor in session with another PLU. See also *controlling application program* and *controlling logical unit*. (2) In VM, a process by which a virtual machine is initiated by other than the user of that virtual machine. For example, the primary VM operator's virtual machine is activated automatically during VM initialization.

available. In VTAM, pertaining to a logical unit that is active, connected, enabled, and not at its session limit.

bidder. In SNA, the LU-LU half-session defined at session activation as having to request and receive permission from the other LU-LU half-session to begin a bracket. Contrast with *first speaker*. See also *bracket protocol* and *contention*.

boundary function. (1) A capability of a subarea node to provide protocol support for attached peripheral nodes, such as: (a) interconnecting subarea path control and peripheral path control elements, (b) performing session sequence numbering for low-function peripheral nodes, and (c) providing session-level pacing support. (2) The component that provides these capabilities. See also *boundary node, network addressable unit (NAU), peripheral path control, subarea node,* and *subarea path control*.

boundary node. (1) A subarea node with boundary function. See *subarea node* (including illustration). See also *boundary function*. (2) The programming component that performs FID2 (format identification type 2) conversion, channel data link control, pacing, and channel or device error recovery procedures for a locally attached station. These functions are similar to those performed by a network control program for an NCP-attached station.

bracket protocol. In SNA, a data flow control protocol in which exchanges between the two LU-LU half-sessions are achieved through the use of brackets, with one LU designated at session activation as the first speaker and the other as the bidder. The bracket protocol involves bracket initiation and termination rules. See also *bidder* and *first speaker*.

branch exchange. A switching system that provides telephone communication between branch stations and external networks.

buffer. A portion of storage for temporarily holding input or output data.

CBX. Computerized branch exchange.

chain. See *RU chain*.

channel. * A path along which signals can be sent, for example, data channel, output channel. See *data channel* and *input/output channel*. See also *link*.

character-coded. Synonym for *unformatted*.

CICS. Customer Information Control System.

CLIST. Command list.

cluster controller. A device that can control the input/output operations of more than one device connected to it. A cluster controller may be controlled by a program stored and executed in the unit; for example, the IBM 3601 Finance Communication Controller. Or it may be controlled entirely by hardware; for example, the IBM 3272 Control Unit.

CNM. Communication network management.

command. (1) A request from a terminal for the performance of an operation or the execution of a particular program. (2) In SNA, any field set in the transmission header (TH), request header (RH), and sometimes portions of a request unit (RU), that initiates an action or that begins a protocol; for example: (a) Bind Session (session-control request unit), a command that activates an LU-LU session, (b) the change-direction indicator in the RH of the last RU of a chain, (c) the virtual route reset window indicator in a FID4 transmission header. See also *VTAM operator command*.

command facility. The component of the NetView program that is a base for command processors that can monitor, control, automate, and improve the operation of a network.

command list (CLIST). In the NetView program, a sequential list of commands and control statements that is assigned a name. When the name is invoked (as a command) the commands in the list are executed.

command processor. A program that performs an operation specified by a command.

communication line. Deprecated term for *telecommunication line* and *transmission line*.

communication management configuration host node. The type 5 host processor in a communication management configuration that does all network-control functions in the network except for the control of devices channel-attached to data hosts. Synonymous with *communication management host*. Contrast with *data host node*.

communication management host. Synonym for *communication management configuration host node*.

communication network management (CNM). The process of designing, installing, operating, and managing the distribution of information and controls among end users of communication systems.

communication network management (CNM) application program. A VTAM application program that issues and receives formatted management services request units for physical units. For example, NetView.

communication network management (CNM) interface. The interface that the access method provides to an application program for handling data and commands associated with communication system management. CNM data and commands are handled across this interface.

communication network management (CNM) processor. A program that manages one of the functions of a communications system. A CNM processor is executed under control of NetView.

composite end node (CEN). A group of nodes made up of a single type 5 node and its subordinate type 4 nodes that together support type 2.1 protocols. To a type 2.1 node, a CEN appears as one end node.

computerized branch exchange (CBX). An exchange in which a central node acts as a high-speed switch to establish direct connections between pairs of attached nodes.

configuration. (1) (TC97) The arrangement of a computer system or network as defined by the nature, number, and the chief characteristics of its functional units. The term may refer to a hardware or a software configuration. (2) The devices and programs that make up a system, subsystem, or network. (3) In CCP, the arrangement of controllers, lines, and terminals attached to an IBM 3710 Network Controller. Also, the collective set of item definitions that describe such a configuration.

configuration services. In SNA, one of the types of network services in the control point (CP) and in the physical unit (PU); configuration services activate, deactivate, and maintain the status of physical units, links, and link stations. Configuration services also shut down and restart network elements and modify path control routing tables and address-translation tables. See also *maintenance services*, *management services*, *network services*, and *session services*.

connected. In VTAM, pertaining to a physical unit (PU) or logical unit (LU) that has an active physical path to the host processor containing the system services control point (SSCP) that controls the PU or LU.

connection. Synonym for *physical connection*.

contention. A situation in which two logical units (LUs) that are connected by an LU 6.2 session both attempt to allocate the session for a conversation at the same time. The control operator assigns "winner" and "loser" status to the LUs so that processing may continue on an orderly basis. The contention loser requests permission from the contention winner to allocate a conversation on the session, and the contention winner either grants or rejects the request. See also *bidder*.

control block. (ISO) A storage area used by a computer program to hold control information.

control point (CP). (1) A system services control point (SSCP) that provides hierarchical control of a group of nodes in a network. (2) A control point (CP) local to a specific node that provides control of that node, either in the absence of SSCP control (for type 2.1 nodes engaged in peer to peer communication) or to supplement SSCP control.

control program (CP). The VM operating system that manages the real processor's resources and is responsible for simulating System/370s for individual users.

controlling application program. In VTAM, an application program with which a secondary logical unit (other than an application program) is automatically put in session whenever the secondary logical unit is available. See also *automatic logon* and *controlling logical unit*.

controlling logical unit. In VTAM, a logical unit with which a secondary logical unit (other than an application program) is automatically put in session whenever the secondary logical unit is available. A controlling logical unit can be either an application program or a device-type logical unit. See also *automatic logon* and *controlling application program*.

CP. (1) Control program. (2) Control point.

Customer Information Control System (CICS). A licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It also includes facilities for building, using, and maintaining data bases.

DASD. Direct access storage device.

data channel. Synonym for *input/output channel*. See *channel*.

data flow control (DFC) layer. In SNA, the layer within a half-session that (1) controls whether the half-session can send, receive, or concurrently send and receive request units (RUs); (2) groups related RUs into RU chains; (3) delimits transactions via the bracket protocol; (4) controls the interlocking of requests and responses in accordance with control modes specified at session activation; (5) generates sequence numbers; and (6) correlates requests and responses.

data host. Synonym for *data host node*.

data host node. In a communication management configuration, a type 5 host node that is dedicated to processing applications and does not control network resources, except for its channel-attached or communication adapter-attached devices. Synonymous with *data host*. Contrast with *communication management configuration host node*.

data link. In SNA, synonym for *link*.

data link control protocol. In SNA, a set of rules used by two nodes on a data link to accomplish an orderly exchange of information. Synonymous with *line control*.

data services command processor (DSCP). A component that structures a request for recording and retrieving data in the application program's data base and for soliciting data from a device in the network.

data services task (DST). The NetView subtask that gathers, records, and manages data in a VSAM file and/or a network device that contains network management information.

data types. In the NetView program, a concept to describe the organization of panels. Data types are defined as alerts, events, and statistics. Data types are combined with resource types and display types to describe NetView's display organization. See also *display types* and *resource types*.

deallocate. A logical unit (LU) 6.2 application program interface (API) verb that terminates a conversation, thereby freeing the session for a future conversation. Contrast with *allocate*.

definite response (DR). In SNA, a value in the form-of-response-requested field of the request header. The value directs the receiver of the request to return a response unconditionally, whether positive or negative, to that request. Contrast with *exception response* and *no response*.

definition statement. (1) In VTAM, the statement that describes an element of the network. (2) In NCP, a type of instruction that defines a resource to the NCP. See Figure 131, Figure 132, and Figure 133 on page 393. See also *macroinstruction*.

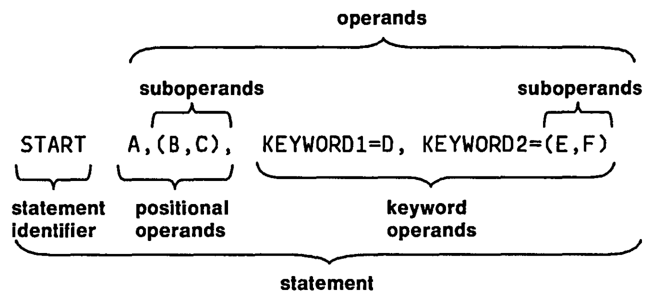


Figure 131. Example of a Language Statement

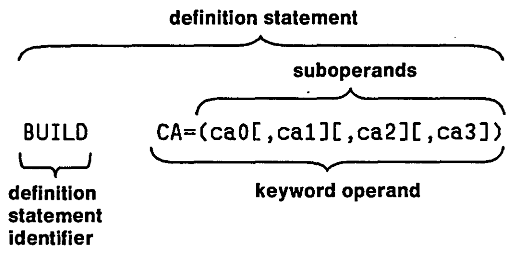

definition statement
suboperands
BUILD CA=(ca0[,ca1][,ca2][,ca3])
definition statement identifier
keyword operand

Figure 132. NCP Example

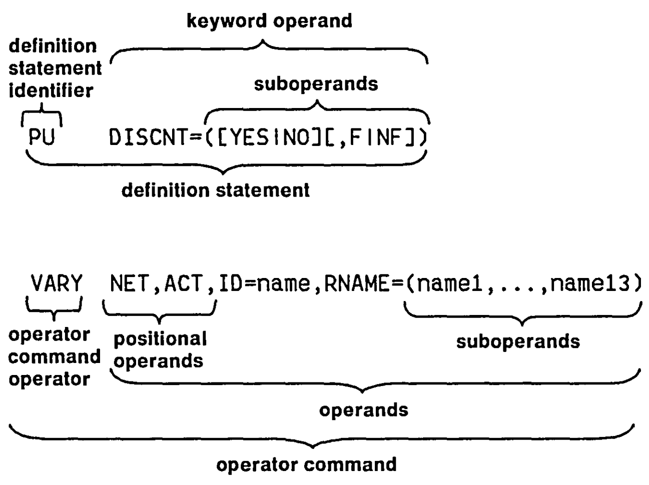


Figure 133. VTAM Examples

detailed data. Short strings of product-specific textual data transported in a network management vector transport (NMVT) and displayed, without any interpretation or translation, by a problem management focal-point product.

direct access storage device (DASD). A device in which the access time is effectively independent of the location of the data. For example, a disk.

directory. In VM, a control program (CP) disk that defines each virtual machine's normal configuration.

disabled. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is temporarily not ready to establish LU-LU sessions. An initiate request for a session with a disabled logical unit (LU) can specify that the session be queued by the SSCP until the LU becomes enabled. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or a secondary logical unit (SLU). See also *enabled* and *inhibited.*

Disk Operating System (DOS). Software for the PC that controls the execution of programs. Its full name is the IBM Personal Computer Disk Operating System.

display. (1) To present information for viewing, usually on a terminal screen or a hard-copy device. (2) A device or medium on which information is presented, such as a terminal screen. (3) Deprecated term for *panel.*

display levels. Synonym for *display types.*

display types. In NetView, a concept to describe the organization of panels. Display types are defined as total, most recent, user action, and detail. Display types are combined with resource types and data types to describe NetView's panel organization. See *data types* and *resource types.* Synonymous with *display levels.*

domain. (1) An access method, its application programs, communication controllers, connecting lines, modems, and attached terminals. (2) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests. See also *single-domain network* and *multiple-domain network.*

domain operator. In a multiple-domain network, the person or program that controls the operation of the resources controlled by one system services control point. Contrast with *network operator* (2).

DOS. Disk Operating System.

DOS partition. In the NetView/PC program, a separate area of memory in which NetView/PC programs and other DOS programs can be serially executed.

downstream. In the direction of data flow from the host to the end user. Contrast with *upstream.*

DSCP. Data services command processor.

DST. Data services task.

dump. (1) Computer printout of storage. (2) To write the contents of all or part of storage to an external medium as a safeguard against errors or in connection with debugging. (3) (ISO) Data that have been dumped.

EBCDIC. * Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

element. (1) A field in the network address. (2) The particular resource within a subarea identified by the element address. See also *subarea.*

enabled. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is now ready to establish LU-LU sessions. The LU can separately indicate whether this prevents it from acting as a primary logical unit (PLU) or as a secondary logical unit (SLU). See also *disabled* and *inhibited.*

end node. A type 2.1 node that does not provide any intermediate routing or session services to any other node. See *composite end node*, *node*, and *type 2.1 node.*

end user. In SNA, the ultimate source or destination of application data flowing through an SNA network. An end user may be an application program or a terminal operator.

ER. (1) Explicit route. (2) Exception response.

error-to-traffic (E/T). The number of temporary errors compared to the traffic associated with a resource.

E/T. Error-to-traffic.

event. (1) In the NetView program, a record indicating irregularities of operation in physical elements of a network. (2) An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as an input/output operation.

exception request (EXR). In SNA, a request that replaces another message unit in which an error has been detected.

exception response (ER). In SNA, a value in the form-of-response-requested field of a request header (RH). An exception response is sent only if a request is unacceptable as received or cannot be processed. Contrast with *definite response* and *no response*. See also *negative response.*

EXEC. In a VM operating system, a user-written command file that contains CMS commands, other user-written commands, and execution control statements, such as branches.

exit routine. Any of several types of special-purpose user-written routines. See *accounting exit routine, authorization exit routine, logon-interpret routine, virtual route selection exit routine, EXLST exit routine*, and *RPL exit routine*.

EXLST exit routine. In VTAM, a routine whose address has been placed in an exit list (EXLST) control block. The addresses are placed there with the EXLST macroinstruction, and the routines are named according to their corresponding operand; hence DFASY exit routine, TPEND exit routine, RELREQ exit routine, and so forth. All exit list routines are coded by the VTAM application programmer. Contrast with *RPL exit routine.*

explicit route (ER). In SNA, the path control network elements, including a specific set of one or more transmission groups, that connect two subarea nodes. An explicit route is identified by an origin subarea address, a destination subarea address, an explicit route number, and a reverse explicit route number. Contrast with *virtual route (VR)*. See also *path* and *route extension.*

EXR. Exception request.

field-formatted. Pertaining to a request or response that is encoded into fields, each having a specified format such as binary codes, bit-significant flags, and symbolic names. Contrast with *character-coded.*

first speaker. In SNA, the LU-LU half-session defined at session activation as: (1) able to begin a bracket without requesting permission from the other LU-LU half-session to do so, and (2) winning contention if both half-sessions attempt to begin a bracket simultaneously. Contrast with *bidder*. See also *bracket protocol.*

focal point. The control point for any management services element containing control of the functions responsible for network management data. See also *management services.*

frame. (1) The unit of transmission in some local area networks, including the IBM Token-Ring Network. It includes delimiters, control characters, information, and checking characters. (2) In SDLC, the vehicle for every command, every response, and all information that is transmitted using SDLC procedures.

full-screen mode. A form of panel presentation in NetView where the contents of an entire terminal screen can be displayed at once. Full-screen mode can be used for fill-in-the-blanks prompting. Contrast with *line mode.*

generation. The process of assembling and link editing definition statements so that resources can be identified to all the necessary programs in a network.

generic alert. A product-independent method of encoding alert data by means of textual data or code points that index short units of stored text.

group. In the NetView/PC program, to identify a set of application programs that are to run concurrently.

half-session. In SNA, a component that provides function management data (FMD) services, data flow control, and transmission control for one of the sessions of a network addressable unit (NAU). See also *primary half-session* and *secondary half-session.*

hardware monitor. The component of the NetView program that helps identify network problems, such as hardware, sotware, and microcode, from a central control point using interactive display techniques.

help desk. In the NetView program, an online information facility that guides the help desk operator through problem management procedures.

help panel. An online display that tells you how to use a command or another aspect of a product. See *task panel.*

hierarchy. In the NetView program, the resource types, display types, and data ty pes that make up the organization, or levels, in a network.

host node. A node providing an application program interface (API) and a common application interface. See *boundary node, network node, node, peripheral* *node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

inactive. Describes the state of a resource that has not been activated or for which the VARY INACT command has been issued. Contrast with *active*. See also *inoperative*.

Information/System. An interactive retrieval program with related utilities designed to provide systems programmers with keyword access to selected technical information contained in either of its companion products, Information/MVS or Information/VM-VSE.

inhibited. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is not ready to establish LU-LU sessions. An initiate request for a session with an inhibited LU will be rejected by the SSCP. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or as a secondary logical unit (SLU). See also *enabled* and *disabled*.

initiate. A network services request sent from a logical unit (LU) to a system services control point (SSCP) requesting that an LU-LU session be established.

inoperative. The condition of a resource that has been active, but is not. The resource may have failed, received an INOP request, or is suspended while a reactivate command is being processed. See also *inactive*.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full screen editor and dialogue manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

interface. * A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs.

ISPF. Interactive System Productivity Facility.

item. In CCP, any of the components, such as communication controllers, lines, cluster controllers, and terminals, that comprise an IBM 3710 Network Controller configuration.

keyword. (1) **(TC97)** A lexical unit that, in certain contexts, characterizes some language construction. (2) * One of the predefined words of an artificial language. (3) One of the significant and informative words in a title or document that describes the content of that document. (4) A name or symbol that identifies a parameter. (5) A part of a command operand that consists of a specific character string (such as DSNAME=). See also *definition statement* and *keyword operand*. Contrast with *positional operand*.

keyword operand. An operand that consists of a keyword followed by one or more values (such as DSNAME=HELLO). See also *definition statement*. Contrast with *positional operand*.

keyword parameter. (1) A parameter that consists of a keyword followed by one or more values.

LCC. Link connection component.

LCSM. Link connection subsystem manager.

line. See *communication line.*

line mode. A form of screen presentation in which the information is presented a line at a time in the message area of the terminal screen. Contrast with *full-screen mode*.

line control. Synonym for *data link control protocol*.

link. In SNA, the combination of the link connection and the link stations joining network nodes; for example: (1) a System/370 channel and its associated protocols, (2) a serial-by-bit connection under the control of Synchronous Data Link Control (SDLC). A link connection is the physical medium of transmission. A link, however, is both logical and physical. Synonymous with *data link*. See Figure 134 on page 396.

link connection. In SNA, the physical equipment providing two-way communication between one link station and one or more other link stations; for example, a telecommunication line and data circuit terminating equipment (DCE).

link connection component (LCC). Components of the link that perform functions for the physical layer of the link.

link connection component manager (LCCM). The transaction program that manages the configuration of the link connection.

link connection segment. A portion of the configuration that is located between two resources listed consecutively in the service point command service (SPCS) query link configuration request list.

link connection subsystem (LCS). The sequence of link connection components (LCCs) that belong to a link connection and are managed by one LCSM.

link connection subsystem manager (LCSM). The transaction program that manages the sequence of link connection components (LCCs) that belong to a link connection.

link station. (1) In SNA, the combination of hardware and software that allows a node to attach to and provide control for a link. (2) In VTAM, a named

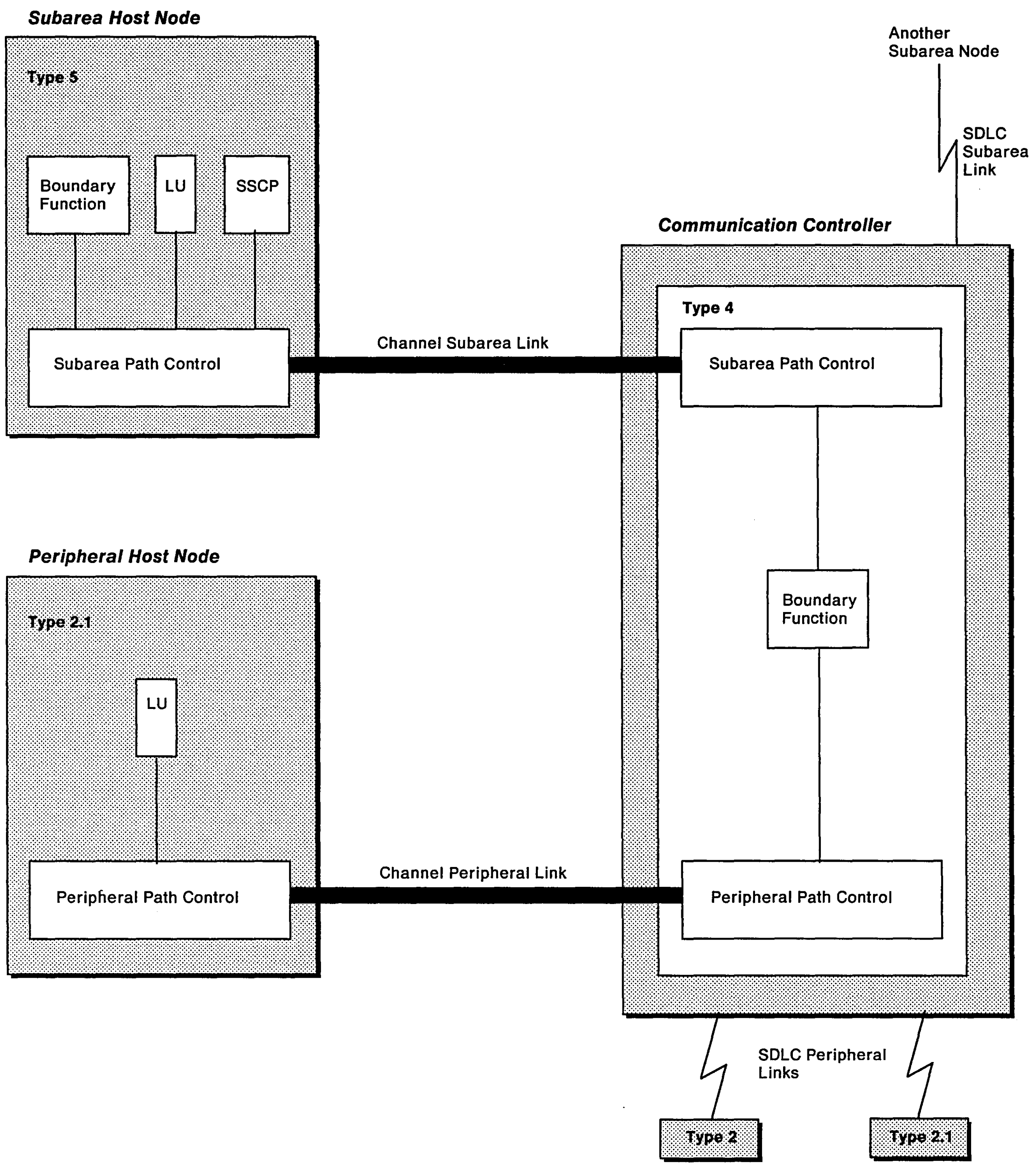


Figure 134. Links and Path Controls

resource within a subarea node that represents another subarea node that is attached by a subarea link. In the resource hierarchy, the link station is subordinate to the subarea link.

link status (LS). Information maintained by local and remote modems.

link test. In SNA, a test in which one link station returns data received from another link station without changing the data in order to test the operation of the link. Three tests can be made; they differ in the resources that are dedicated during the test.

local address. In SNA, an address used in a peripheral node in place of an SNA network address and transformed to or from an SNA network address by the boundary function in a subarea node.

logon. In VTAM, an unformatted session initiation request for a session between two logical units. See *automatic logon* and *simulated logon*. See also *session-initiation request*.

logon-interpret routine. In VTAM, an installation exit routine, associated with an interpret table entry, that translates logon information. It may also verify the logon.

low-entry networking. In SNA, a capability in type 2.1 nodes allowing them to be directly attached to one another (not involving the subarea network) using peer-to-peer protocols and allowing them to support multiple and parallel sessions between logical units (LUs).

LU type. In SNA, the classification of an LU-LU session in terms of the specific subset of SNA protocols and options supported by the logical units (LUs) for that session, namely:

> The mandatory and optional values allowed in the session activation request.
>
> The usage of data stream controls, function management headers (FMHs), request unit (RU) parameters, and sense codes.
>
> Presentation services protocols such as those associated with FMH usage.

LU types 0, 1, 2, 3, 4, 6.1, 6.2, and 7 are defined.

LU-LU session. In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two end users, or between an end user and an LU services component.

LU-LU session type. A deprecated term for *LU type*.

macroinstruction. (1) An instruction that when executed causes the execution of a predefined sequence of instructions in the same source language. (2) In assembler programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macro definition. The statements normally produced from the macro definition replace the macroinstruction in the program. See also *definition statement*.

maintenance services. In SNA, one of the types of network services in system services control points (SSCPs) and physical units (PUs). Maintenance services provide facilities for testing links and nodes and for collecting and recording error information. See also *configuration services, management services, network services*, and *session services*.

major node. In VTAM, a set of resources that can be activated and deactivated as a group. See *node* and *minor node*.

management services. In SNA, one of the types of network services in control points (CPs) and physical units (PUs). Management services are the services provided to assist in the management of SNA networks, such as problem management, performance and accounting management, and charge management. See also *configuration services, maintenance services, network services*, and *session services*.

message. (1) (TC97) A group of characters and control bit sequences transferred as an entity. (2) In VTAM, the amount of function management data (FMD) transferred to VTAM by the application program with one SEND request.

minor node. In VTAM, a uniquely-defined resource within a major node. See *node* and *major node*.

modem. A device that modulates and demodulates signals transmitted over data communication facilities. The term is a contraction for modulator-demodulator.

multiple-domain network. In SNA, a network with more than one system services control point (SSCP). Contrast with *single-domain network*.

Multiple Virtual Storage (MVS). An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for System/370. It is a software operating system controlling the execution of programs.

MVS. Multiple Virtual Storage operating system.

NAU. Network addressable unit.

NC. Network control.

NCCF. Network Communications Control Facility.

NCP. (1) Network Control Program (IBM licensed program). Its full name is Advanced Communications Function for the Network Control Program. Synonymous with *ACF/NCP*. (2) Network control program (general term).

negative response (NR). In SNA, a response indicating that a request did not arrive successfully or was not processed successfully by the receiver. Contrast with *positive response*. See *exception response*.

NetView. A system 370-based IBM licensed program used to monitor a network, manage it, and diagnose its problems.

NetView-NetView task (NNT). The task under which a cross-domain NetView operator session runs. See *operator station tast*.

NetView/PC. A PC-based IBM licensed program through which application programs can be used to monitor, manage, and diagnose problems in IBM Token-Ring networks, non-SNA communication devices, and voice networks.

network. (1) (TC97) An interconnected group of nodes. (2) In data processing, a user application network. See *path control network, public network, SNA network, subarea network, type 2.1 network*, and *user-application network*.

network address. In SNA, an address, consisting of subarea and element fields, that identifies a link, a link station, or a network addressable unit. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See *local address*. See also *network name*.

network addressable unit (NAU). In SNA, a logical unit, a physical unit, or a system services control point. It is the origin or the destination of information transmitted by the path control network. Each NAU has a network address that represents it to the path control network. See also *network name, network address*, and *path control network*.

Network Communications Control Facility (NCCF). (1) An IBM licensed program that is a base for command processors that can monitor, control, automate, and improve the operations of a network. Its function is included and enhanced in NetView's command facility. (2) A traditional, alternative name for the command facility of NetView.

network control (NC). In SNA, an RU category used for requests and responses exchanged for such purposes as activating and deactivating explicit and virtual routes and sending load modules to adjacent peripheral nodes. See also *data flow control layer* and *session control*.

Network Control Program (NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Its full name is Advanced Communications Function for the Network Control Program.

network control program. A program, generated by the user from a library of IBM-supplied modules, that controls the operation of a communication controller.

network management vector transport (NMVT). A record that contains solicited or unsolicited data about alerts, line statistics, and error records and that is issued by certain SNA resources to the host system. It can also be used to send requests on Link Problem Determination Aid (LPDA) lines for certain actions such as configuration changes.

network name. (1) In SNA, the symbolic identifier by which end users refer to a network addressable unit (NAU), a link, or a link station. See also *network address*. (2) In a multiple-domain network, the name of the APPL statement defining a VTAM application program is its network name and it must be unique across domains. Contrast with *ACB name*. See *uninterpreted name*.

network node. (1) Synonym for *type 2.1 node*. Contrast with *end node*. (2) Synonym for *node*.

network operator. (1) A person or program responsible for controlling the operation of all or part of a network. (2) The person or program that controls all the domains in a multiple-domain network. Contrast with *domain operator*.

network services (NS). In SNA, the services within network addressable units (NAUs) that control network operation through SSCP-SSCP, SSCP-PU, and SSCP-LU sessions. See *configuration services, maintenance services, management services*, and *session services*.

network services (NS) header. In SNA, a 3-byte field in a function management data (FMD) request/response unit (RU) flowing in an SSCP-LU, SSCP-PU, or SSCP-SSCP session. The network services header is used primarily to identify the network services category of the request unit (RU) (for example, configuration services, session services) and the particular request code within a category.

NMVT. Network management vector transport.

node. (1) In SNA, an endpoint of a link or junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities. Synonymous with *network node*. See *boundary node, host node, peripheral node*, and *subarea node* (including illustration). (2) In VTAM, a point in a network defined by a symbolic name. See *major node* and *minor node*.

node type. In SNA, a designation of a node according to the protocols it supports and the network addressable units (NAUs) that it can contain. Five types are defined: 1, 2.0, 2.1, 4, and 5. Type 1, type 2.0, and type 2.1 nodes are peripheral nodes; type 4 and type 5 nodes are subarea nodes. See *physical unit type*. See also *type 2.1 node*.

no response. In SNA, a value in the form-of-response-requested field of the request header (RH) indicating that no response is to be returned to the request, whether or not the request is received and processed successfully. Contrast with *definite response* and *exception response*.

notify. A network services request that is sent by an SSCP to a logical unit (LU) to inform the LU of the status of a procedure requested by the LU.

NS. Network services.

online. Stored in a computer and accessible from a terminal.

operand. (1) **(ISO)** An entity on which an operation is performed. (2) * That which is operated upon. An operand is usually identified by an address part of an instruction. (3) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor. (4) An expression to whose value an operator is applied. See also *definition statement*, *keyword*, *keyword parameter*, and *parameter*.

operator. (1) In a language statement, the lexical entity that indicates the action to be performed on operands. (2) A person who operates a machine. See *network operator*. See also *definition statement*.

operator profile. In the NetView program, the resources and activities a network operator has control over. The statements defining these resources and activities are stored in a file that is activated when the operator logs on.

operator station task (OST). The NetView task that establishes and maintains the online session with the network operator. There is one operator station task for each network operator who logs on to NetView. See *NetView-NetView task*.

OST. Operator station task.

pacing group. In SNA, (1) The path information units (PIUs) that can be transmitted on a virtual route before a virtual-route pacing response is received, indicating that the virtual route receiver is ready to receive more PIUs on the route. Synonymous with *window*. (2) The requests that can be transmitted on the normal flow in one direction on a session before a session-level pacing response is received, indicating that the receiver is ready to accept the next group of requests.

page. (1) The portion of a panel that is shown on a display surface at one time. (2) To move back and forth among the pages of a multiple-page panel. See also *scroll*. (3) (ISO) In a virtual storage system, a fixed-length block that has a virtual address and that can be transferred between real storage and auxiliary storage. (4) To transfer instructions, data, or both between real storage and external page or auxiliary storage.

panel. (1) A formatted display of information that appears on a terminal screen. See also *help panel* and *task panel*. Contrast with *screen*. (2) In computer graphics, a display image that defines the locations and characteristics of display fields on a display surface.

parameter. (1) **(ISO)** A variable that is given a constant value for a specified application and that may denote the application. (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (3) Data passed to a program or procedure by a user or another program, namely as an operand in a language statement, as an item in a menu, or as a shared data structure. See also *keyword*, *keyword parameter*, and *operand*.

path. (1) In SNA, the series of path control network components (path control and data link control) that are traversed by the information exchanged between two network addressable units (NAUs). See also *explicit route (ER)*, *route extension*, and *virtual route (VR)*.
(2) In VTAM when defining a switched major node, a potential dial-out port that can be used to reach that node. (3) In the NetView/PC program, a complete line in a configuration that contains all of the resources in the service point command service (SPCS) query link configuration request list.

path control (PC). The function that routes message units between network addressable units (NAUs) in the network and provides the paths between them. It converts the BIUs from transmission control (possibly segmenting them) into path information units (PIUs) and exchanges basic transmission units (BTUs) and one or more PIUs with data link control. Path control differs for peripheral nodes, which use local addresses for routing, and subarea nodes, which use network addresses for routing. See *peripheral path control* and *subarea path control*. See also *link*, *peripheral node*, and *subarea node*.

path control (PC) layer. In SNA, the layer that manages the sharing of link resources of the SNA network and routes basic information units (BIUs) through it. See also *BIU segment*, *blocking of PIUs*, *data link control layer*, and *transmission control layer*.

path control (PC) network. In SNA, the part of the SNA network that includes the data link control and path control layers. See *SNA network* and *user application network*. See also *boundary function*.

PBX. Private branch exchange.

PC. (1) Path control. (2) Personal Computer. Its full name is the IBM Personal Computer.

peripheral host node. A node that provides an application program interface (API) for running application programs but does not provide SSCP functions and is not aware of the network configuration. The peripheral host node does not provide subarea node services. It has boundary function provided by its adjacent subarea. See *boundary node*, *host node*, *network* *node*, *node*, *peripheral node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

peripheral node. In SNA, a node that uses local addresses for routing and therefore is not affected by changes in network addresses. A peripheral node requires boundary-function assistance from an adjacent subarea node. A peripheral node is a physical unit (PU) type 1, 2.0, or 2.1 node connected to a subarea node with boundary function within a subarea. See *boundary node*, *host node*, *network node*, *node*, *peripheral host node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type.*

peripheral path control. The function in a peripheral node that routes message units between units with local addresses and provides the paths between them. See *path control* and *subarea path control.* See also *boundary function*, *peripheral node*, and *subarea node*.

peripheral PU. In SNA, a physical unit representing a peripheral node.

Personal Computer (PC). The IBM Personal Computer line of products including the 5150 and subsequent models.

physical connection. In VTAM, a point-to-point connection or multipoint connection. Synonymous with *connection*.

physical unit (PU). In SNA, a type of network addressable unit (NAU). A physical unit (PU) manages and monitors the resources (such as attached links) of a node, as requested by a system services control point (SSCP) through an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links. See also *peripheral PU* and *subarea PU*.

physical unit (PU) services. In SNA, the components within a physical unit (PU) that provide configuration services and maintenance services for SSCP-PU sessions. See also *logical unit (LU) services.*

PLU. Primary logical unit.

POI. Programmed operator interface.

polling. (1) * Interrogation of devices for purposes such as to avoid contention, to determine operational status, or to determine readiness to send or receive data. (2) (TC97) The process whereby stations are invited, one at a time, to transmit.

positional operand. An operand in a language statement that has a fixed position. See also *definition statement.* Contrast with *keyword operand.*

positive response. A response indicating that a request was received and processed. Contrast with *negative response*.

presentation services command processor (PSCP). In NetView, a facility that processes requests from a user terminal and formats displays to be presented at the user terminal.

primary half-session. In SNA, the half-session that sends the session activation request. See also *primary logical unit.* Contrast with *secondary half-session.*

primary logical unit (PLU). In SNA, the logical unit (LU) that contains the primary half-session for a particular LU-LU session. Each session must have a PLU and secondary logical unit (SLU). The PLU is the unit responsible for the bind and is the controlling LU for the session. A particular LU may contain both primary and secondary half-sessions for different active LU-LU sessions. Contrast with *secondary logical unit (SLU).*

private branch exchange. A switching system that provides internal telephone communication between private branch stations and external networks.

problem determination. The process of identifying the source of a problem; for example, a program component, a machine failure, telecommunication facilities, user or contractor-installed programs or equipment, an environment failure such as a power loss, or a user error.

product-set identification (PSID). (1) In SNA, a technique for identifying the hardware and software products that implement a network component. (2) A management services common subvector that transports the information described in definition (1).

profile. In the Conversational Monitor System (CMS) or the group control system (GCS), the characteristics defined by a PROFILE EXEC file that executes automatically after the system is loaded into a virtual machine. See also *operator profile.*

programmed operator interface (POI). A VTAM function that allows programs to perform VTAM operator functions.

protocol. (1) (CCITT/ITU) A specification for the format and relative timing of information exchanged between communicating parties. (2) (TC97) The set of rules governing the operation of functional units of a communication system that must be followed if communication is to be achieved. (3) In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components. See also *bracket protocol.* Synonymous with *line control discipline* and *line discipline.* See also *link protocol.*

PSCP. Presentation services command processor.

PSID. Product-set identification.

PU. Physical unit.

public network. A network established and operated by communication common carriers or telecommunication Administrations for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public. Contrast with *user-application network*.

PU-PU flow. In SNA, the exchange between physical units (PUs) of network control requests and responses.

RECFMS. Record formatted maintenance statistics.

Recommendation X.21 (Geneva 1980). A Consultative Committee on International Telegraph and Telephone (CCITT) recommendation for a general purpose interface between data terminal equipment and data circuit equipment for synchronous operations on a public data network.

Recommendation X.25 (Geneva 1980). A Consultative Committee on International Telegraph and Telephone (CCITT) recommendation for the interface between data terminal equipment and packet-switched data networks. See also *packet switching*.

recommended action. Procedures suggested by NetView that can be used to determine the causes of network problems.

record formatted maintenance statistics (RECFMS). A statistical record built by an SNA controller and usually solicited by the host.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks. For example, the 3710 Network Controller routines may be reentrant.

release. For VTAM, to relinquish control of resources (communication controllers or physical units). See also *resource takeover*. Contrast with *acquire (2)*.

remote. Concerning the peripheral parts of a network not centrally linked to the host processor and generally using telecommunication lines with public right-of-way.

REQMS. Request for maintenance statistics.

request for maintenance statistics (REQMS). A host solicitation to an SNA controller for a statistical data record.

request unit (RU). In SNA, a message unit that contains control information, end-user data, or both.

request/response unit (RU). In SNA, a generic term for a request unit or a response unit. See also *request unit (RU)* and *response unit*.

resource. (1) Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs. (2) In the NetView program, any hardware or software that provides function to the network.

resource takeover. In VTAM, action initiated by a network operator to transfer control of resources from one domain to another. See also *acquire (2)* and *release*. See *takeover*.

resource types. In the NetView program, a concept to describe the organization of panels. Resource types are defined as central processing unit, channel, control unit, and I/O device for one category; and communication controller, adapter, link, cluster controller, and terminal for another category. Resource types are combined with data types and display types to describe display organization. See also *data types* and *display types*.

response time. (1) The amount of time it takes after a user presses the enter key at the terminal until the reply appears at the terminal. (2) For response time monitoring, the time from the activation of a transaction until a response is received, according to the response time definition coded in the performance class.

response unit (RU). In SNA, a message unit that acknowledges a request unit; it may contain prefix information received in a request unit. If positive, the response unit may contain additional information (such as session parameters in response to Bind Session), or if negative, contains sense data defining the exception condition.

return code. * A code [returned from a program] used to influence the execution of succeeding instructions.

ring. A network configuration where a series of attaching devices are connected by unidirectional transmission links to form a closed path.

route extension (REX). In SNA, the path control network components, including a peripheral link, that make up the portion of a path between a subarea node and a network addressable unit (NAU) in an adjacent peripheral node. See also *path, explicit route (ER),* and *virtual route (VR)*.

RPL exit routine. In VTAM, an application program exit routine whose address has been placed in the EXIT field of a request parameter list (RPL). VTAM invokes the routine to indicate that an asynchronous request has been completed. See *EXLST exit routine*.

RU. Request/response unit.

RU chain. In SNA, a set of related request/response units (RUs) that are consecutively transmitted on a particular normal or expedited data flow. The request RU chain is the unit of recovery: if one of the RUs in the chain cannot be processed, the entire chain is discarded. Each RU belongs to only one chain, which has a beginning and an end indicated by means of control bits in request/response headers within the RU chain. Each RU can be designated as first-in-chain (FIC), last-in-chain (LIC), middle-in-chain (MIC), or only-in-chain (OIC). Response units and expedited-flow request units are always sent as only-in-chain.

SC. Session control.

screen. An illuminated display surface; for example, the display surface of a CRT or plasma panel. Contrast with *panel*.

scroll. To move all or part of the display image vertically to display data that cannot be observed within a single display image. See also *page (2)*.

SDLC. Synchronous Data Link Control.

secondary half-session. In SNA, the half-session that receives the session-activation request. See also *secondary logical unit (SLU)*. Contrast with *primary half-session*.

secondary logical unit (SLU). In SNA, the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. An LU may contain secondary and primary half-sessions for different active LU-LU sessions. Contrast with *primary logical unit (PLU)*.

secondary logical unit (SLU) key. A key-encrypting key used to protect a session cryptography key during its transmission to the secondary half-session.

segment. See *link connection segment*.

Service Level Reporter (SLR). A licensed program that generates management reports from data sets such as System Management Facility (SMF) files.

service point (SP). A control point that provides network management to non-SNA devices.

service point command facility (SPCF). A program or function that exchanges data and control between the network operator, the link connection component manager (LCCM), and the link connection subsystem manager (LCSM).

service reminder (SR). In the NetView/PC program, a notification set by the operator that is displayed on a panel and logs a specified message.

session. In SNA, a logical connection between two network addressable units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. Each session is uniquely identified in a transmission header (TH) by a pair of network addresses, identifying the origin and destination NAUs of any transmissions exchanged during the session. See *half-session*, *LU-LU session*, *SSCP-LU session*, *SSCP-PU session*, and *SSCP-SSCP session*. See also *LU-LU session type* and *PU-PU flow*.

session awareness (SAW) data. Data collected by NetView about a session that includes the session type, the names of session partners, and information about the session activation status. It is collected for LU-LU, SSCP-LU, SSCP-PU, and SSCP-SSCP sessions and for non-SNA terminals not supported by NTO. It can be displayed in various forms, such as most recent sessions lists.

session control (SC). In SNA, (1) One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification. (2) A request unit (RU) category used for requests and responses exchanged between the session control components of a session and for session activation and deactivation requests and responses.

session-initiation request. In SNA, an Initiate or logon request from a logical unit (LU) to a control point (CP) that an LU-LU session be activated.

session monitor. The component of NetView that collects and correlates session-related data and provides online access to this information.

session services. In SNA, one of the types of network services in the control point (CP) and in the logical unit (LU). These services provide facilities for an LU or a network operator to request that the SSCP initiate or terminate sessions between logical units. See *configuration services*, *maintenance services*, and *management services*.

shared. Pertaining to the availability of a resource to more than one use at the same time.

shutdown. To stop or quiesce a NetView/PC or a NetView/PC application program.

simulated logon. A session-initiation request generated when a VTAM application program issues a SIMLOGON macroinstruction. The request specifies a logical unit (LU) with which the application program wants a session in which the requesting application program will act as the primary logical unit (PLU).

single-domain network. In SNA, a network with one system services control point (SSCP). Contrast with *multiple-domain network*.

SLR. Service Level Reporter.

SLU. Secondary logical unit.

SNA. Systems Network Architecture.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network addressable units (NAUs), boundary function components, and the path control network.

solicited message. A response from VTAM to a command entered by a program operator. Contrast with *unsolicited message.*

SP. Service point.

SPCF. Service point command facility.

SR. Service reminder.

SS. Start-stop.

SSCP. System services control point.

SSCP-LU session. In SNA, a session between a system services control point (SSCP) and a logical unit (LU); the session enables the LU to request the SSCP to help initiate LU-LU sessions.

SSCP-PU session. In SNA, a session between a system services control point (SSCP) and a physical unit (PU); SSCP-PU sessions allow SSCPs to send requests to and receive status information from individual nodes in order to control the network configuration.

SSCP-SSCP session. In SNA, a session between the system services control point (SSCP) in one domain and the SSCP in another domain. An SSCP-SSCP session is used to initiate and terminate cross-domain LU-LU sessions.

ST. Session configuration screen abbreviation.

statement. A language syntactic unit consisting of an operator, or other statement identifier, followed by one or more operands. See *definition statement.*

station. (1) One of the input or output points of a network that uses communication facilities; for example, the telephone set in the telephone system or the point where the business machine interfaces with the channel on a leased private line. (2) One or more computers, terminals, or devices at a particular location.

subarea. A portion of the SNA network consisting of a subarea node, any attached peripheral nodes, and their associated resources. Within a subarea node, all network addressable units, links, and adjacent link stations (in attached peripheral or subarea nodes) that are addressable within the subarea share a common subarea address and have distinct element addresses.

subarea host node. A host node that provides both subarea function and an application program interface (API) for running application programs. It provides system services control point (SSCP) functions, subarea node services, and is aware of the network configuration. See *boundary node*, *communication management configuration host node*, *data host node*, *host node*, *network node*, *node*, *peripheral node*, and *subarea node*. See also *boundary function* and *node type*.

subarea node. In SNA, a node that uses network addresses for routing and whose routing tables are therefore affected by changes in the configuration of the network. Subarea nodes can provide gateway function, and boundary function support for peripheral nodes. Type 4 and type 5 nodes are subarea nodes. See *boundary node*, *host node*, *network node*, *node*, *peripheral node*, and *subarea host node*. See also *boundary function* and *node type*.

subarea path control. The function in a subarea node that routes message units between network addressable units (NAUs) and provides the paths between them. See *path control* and *peripheral path control*. See also *boundary function*, *peripheral node*, and *subarea node*.

subarea PU. In SNA, a physical unit (PU) in a subarea node.

subsystem. A secondary or subordinate system, usually capable of operating independent of, or asynchronously with, a controlling system.

Synchronous Data Link Control (SDLC). A discipline for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop. SDLC conforms to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute and High-Level Data Link Control (HDLC) of the International Standards Organization.

system services control point (SSCP). In SNA, a central location point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network. Multiple SSCPs, cooperating as peers, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its domain.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

System Support Programs (SSP). An IBM licensed program, made up of a collection of utilities and small programs, that supports the operation of the NCP.

takeover. The process by which the failing active subsystem is released from its extended recovery facility (XRF) sessions with terminal users and replaced by an alternate subsystem. See *resource takeover*.

task. A basic unit of work to be accomplished by a computer. The task is usually specified to a control program in a multiprogramming or multiprocessing environment.

task panel. Online display from which you communicate with the program in order to accomplish the program's function, either by selecting an option provided on the panel or by entering an explicit command. See *help panel*.

telecommunication line. Any physical medium such as a wire or microwave beam, that is used to transmit data. Synonymous with *transmission line*.

terminal. A device that is capable of sending and receiving information over a link; it is usually equipped with a keyboard and some kind of display, such as a screen or a printer.

TERMINATE. In SNA, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure to end one or more designated LU-LU sessions.

TH. Transmission header.

threshold. In the NetView program, refers to a percentage value set for a resource and compared to a calculated error-to-traffic ratio.

token. A sequence of bits passed from one device to another along the token ring. When the token has data appended to it, it becomes a frame.

token ring. A network with a ring topology that passes tokens from one attaching device to another. For example, the IBM Token-Ring Network.

transmission header (TH). In SNA, control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to route message units and to control their flow within the network. See also *path information unit*.

transmission line. Synonym for *telecommunication line*.

tutorial. Online information presented in a teaching format.

type 2.1 node (T2.1 node). A node that can attach to an SNA network as a peripheral node using the same protocols as type 2.0 nodes. Type 2.1 nodes can be directly attached to one another using low-entry networking. Synonymous with *network node*. See *end node*, *node*, and *subarea node*. See also *node type* and *low-entry networking*.

unformatted. In VTAM, pertaining to commands (such as LOGON or LOGOFF) entered by an end user and sent by a logical unit in character form. The character-coded command must be in the syntax defined in the user's unformatted system services definition table. Synonymous with *character-coded*. Contrast with *field-formatted*.

uninterpreted name. In SNA, a character string that a system services control point (SSCP) is able to convert into the network name of a logical unit (LU). Typically, an uninterpreted name is used in a logon or Initiate request from a secondary logical unit (SLU) to identify the primary logical unit (PLU) with which the session is requested.

unsolicited message. A message, from VTAM to a program operator, that is unrelated to any command entered by the program operator. Contrast with *solicited message*.

upstream. In the direction of data flow from the end user to the host. Contrast with *downstream*.

user. Anyone who requires the services of a computing system.

user-application network. A configuration of data processing products, such as processors, controllers, and terminals, established and operated by users for the purpose of data processing or information exchange, which may use services offered by communication common carriers or telecommunication Administrations. Contrast with *public network*.

using node. (1) In NCP, the NCP in the hosts's domain that reports a link error condition. (2) For the command facility of NetView and for NCCF, the ID parameter of certain network control commands.

value. (1) **(TC97)** A specific occurence of an attribute, for example, "blue" for the attribute "color." (2) A quantity assigned to a constant, a variable, a parameter, or a symbol.

variable. In the NetView program, a character string beginning with & that is coded in a command list and is assigned a value during execution of the command list.

verb. (1) In SNA, the general name for a transaction program's request for communication services. (2) In

VTAM, a programming language element in the logical unit (LU) 6.2 application program interface (API) that causes an LU 6.2 function to be performed.

Virtual Machine (VM). A licensed program whose full name is the Virtual Machine/System Product (VM/SP). It is a software operating system that manages the resources of a real processor to provide virtual machines to end users. As a time-sharing system control program, it consists of the virtual machine control program (CP), the conversational monitor system (CMS), the group control system (GCS), and the interactive problem control system (IPCS).

virtual route (VR). In SNA, a logical connection (1) between two subarea nodes that is physically realized as a particular explicit route, or (2) that is contained wholly within a subarea node for intranode sessions. A virtual route between distinct subarea nodes imposes a transmission priority on the underlying explicit route, provides flow control through virtual-route pacing, and provides data integrity through sequence numbering of path information units (PIUs). See also *explicit route (ER)*, *path*, and *route extension*.

virtual route (VR) pacing. In SNA, a flow control technique used by the virtual route control component of path control at each end of a virtual route to control the rate at which path information units (PIUs) flow over the virtual route. VR pacing can be adjusted according to traffic congestion in any of the nodes along the route. See also *pacing* and *session-level pacing*.

virtual route selection exit routine. In VTAM, an optional installation exit routine that modifies the list of virtual routes associated with a particular class of service before a route is selected for a requested LU-LU session.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VM. Virtual Machine operating system. Its full name is Virtual Machine/System Product. Synonymous with *VM/SP*.

VR. Virtual route.

VTAM. Virtual Telecommunications Access Method (IBM licensed program). Its full name is Advanced Communications Function for the Virtual Telecommunications Access Method. Synonymous with *ACF/VTAM*.

VTAM operator command. A command used to monitor or control a VTAM domain. See also *definition statement*.

window. (1) In SNA, synonym for *pacing group*. (2) On a visual display terminal, a small amount of information in a framed-in area on a panel that overlays part of the panel.

X.21. See *Recommendation X.21 (Geneva 1980)*.

Index

A
Alert calls 25
ALERT Description records 84
Alert Router 25
ALERT SV X'9F' Code Point File : DUPALGTF.TXT 83
Alert Type 78
Alert Type records 84
Alerts 9
Application Request Block address in the AX and DX register pair 18
ARBID 10
ASCII 18, 36, 72
ASCII upper case. 42
AX 18
AX register 18

B
BIOS 17
blkz 58

C
cause code 79
cause code point 80
Cause subfield 80
Check the status of a Host Data Facility request 61
Clear the icon from the NetView/PC icon window 33
close 10
Close the Alert API/CS 30
Close the Host Data Facility API/CS 62
Close the Operator Communications API/CS 33
Close the SPCF API/CS 45
code design (application mainline) 12
Code Point File : DUPALGTF.TXT 83
code segment 10
common fields 79
Communications Network Management 20
correlator 36

D
Data Services Task 35
DDM 22
declarations 10
DELAYED 79
DOS Applications 21
DP 31
DUPALGCF.TXT 79, 80, 83
DX 18
DX register 18

E
EBCDIC 36, 72
editor 36
EXTERN 10
external declarations 10
eye catcher 10
EZ–VU 18, 36

F
Failure Cause Subfield 81
Failure Cause type records 86
FAR call 18
flow (application) 12
Functions 9

H
HDFState 58
hexadecimal 10
HFLGTH 58
Host Data Facility 9, 11
HOSTFILE 58
hybrid alert 77

I
icon 31
initialization 10, 12
Install Cause Subfield 81
Install Cause type records 86
interrupt vectors 17

L
language 9
library 10, 18
link 17, 18
LINK—CHANGE 107
LINK—DISPLAY 108
LINKCHNG 107
LINKDATA command
  description 89
  syntax 89
LINKDISP 108
LINKPD command
  description 91
  syntax 91
LINKTEST command
  description 92
  syntax 92

M
Message Data 42
message files 36
Msgcount 42
MVADDR 26
MVTARG 26

N
NetView 25, 35, 36
NetView/PC Alert Description Code Point. 80
NetView/PC Alert Major Vector 80
NetView/PC ALERT SV X'9F' Code Point File :
 DUPALGTF.TXT 83
NetView/PC generic alert example 88
NetView/PC Non-generic Alert example 78
Nextbyte 58
NMVT 18, 25
Non-generic Alerts 77

O
Object Modules 18
offset address 18
open 10
Open Communication Architectures 20
Open the alert API/CS 30
Open the Operator Communications API/CS 32
Open the SPCF API/CS 40
Operator Communications 9, 11, 31
Operator Services Task 35

P
PATH—CHANGE 111
PATH—DISPLAY 110
PATHCHNG 111
PATHDISP 110
PCFILE 58
PCFLGTH 58
PERM ERROR 79
Probable Cause Subfield 81
Probable Cause Type records 84
Problems 9

R
Receive a RUNCMD message 40
Receive file data 61
recommended action code point 80
Recommended Action type records 86
Recvcorr 36
request code 10
Requirements 17
RESCHNG 109
RESDISP 108
RESOURCE—CHANGE 109
RESOURCE—DISPLAY 108
Restrictions 17
RUNCMD command
 description 93
 syntax 93
RUNCMD response message 36
RUNCMD Response Message Buffer 42

S
segment address 18
Send a Command Response 43
Send a RUNCMD response 41
Send an Alert 30
Send file data 60
SENDCORR 36
Service Point Command Facility 9, 11, 35
SPCF Build and Parse 9
SPCF NMVT Header 95
specific component 80
STACK 17
Start byte 58
Stop file data transfer 61
storage 10
subdirectory 36

T
termination 10, 12
transfer 57
translate 25, 36
translate (table) 72
transparent 58

U
User Cause Subfield 81
User Cause type records 85

V
vectors (interrupt) 17

W
Write the icon 'DP' to the NetView/PC icon window 32

X
xpc 58

Numerics
9F 80

Reader's Comment Form

NetView /PC™
Application Program Interface/
Communication Services
Version 1.1

Publication No. SC30-3313-1

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

If you wish a reply, give your name, company, mailing address, and date:

Comments:

What is your occupation?

Number of latest Newsletter associated with this publication:

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC30-3313-1

Reader's Comment Form

Fold and tape **Please Do Not Staple** **Fold and tape**

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E53
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape **Please Do Not Staple** **Fold and tape**

IBM®

Reader's Comment Form

NetView /PC™
Application Program Interface/
Communication Services
Version 1.1

Publication No. SC30-3313-1

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

Comments: ______________________________

____

____

____

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ______________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC30-3313-1

Reader's Comment Form

Fold and tape **Please Do Not Staple** **Fold and tape**

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E53
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape **Please Do Not Staple** **Fold and tape**

IBM®

IBM®

Publication Number
SC30-3313-1

File Number
S370/4300/30XX-50

Program Number
5669-024

Printed in USA

SC30-3313-01