

REFERENCE MANUAL



NaturalLink Window Manager

Part No. 2240317-0001 *C
August 1985

TEXAS INSTRUMENTS

A decorative footer graphic consisting of three horizontal lines. The top two lines are thin and dark brown, while the bottom line is a thick, solid dark brown bar that spans the width of the page.

MANUAL REVISION HISTORY

NaturalLink™ Window Manager Reference Manual (2240317-0001)

Original Issue.....December 1983

Revision.....April 1984

Revision.....November 1984

RevisionAugust 1985

© 1983, 1984, 1985, Texas Instruments Incorporated. All Rights Reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Texas Instruments Incorporated.

The computers, as well as the programs that TI has created to use with them, are tools that can help people better manage the information used in their business; but tools—including TI computers—cannot replace sound judgment nor make the manager's business decisions.

Consequently, TI cannot warrant that its systems are suitable for any specific customer application. The manager must rely on judgment of what is best for his or her business.

The system-defined windows shown in this manual are examples of the software as this manual goes into production. Later changes in the software may cause the windows on your system to be different from those in the manual.

ABOUT THIS MANUAL

This reference manual describes the NaturalLink™ Window Manager and its associated utilities: Screen Builder, Message Builder, and Message Manager. It also provides an overview of screen management and explains how to use it.

This manual includes an index and is organized into the following sections and appendixes.

Chapter 1: Introduction — Introduces the concept of a menu-based, interactive interface for the computer user and describes the major components and capabilities of the Window Manager and Screen Builder utilities.

Chapter 2: Window Manager Features — Provides explanations of the formats, attributes, and features associated with windows and items within windows.

Chapter 3: Window Attributes — Discusses the various attributes that can be assigned to windows and items and describes their characteristics.

Chapter 4: Screen Builder — Discusses the Screen Builder utility. This section also lists the Screen Builder commands and explains how they are used.

Chapter 5: Message Builder — Discusses the Message Builder and Message Manager utilities used to create and display text messages.

Chapter 6: Window Manager Callable Routines — Lists the callable routines that invoke Window Manager, describes the nature of each routine, and explains the effect each call has on your windows.

Chapter 7: Application Validation Routine — Explains why and when edit field validation is done and how Window Manager uses the application validation routine.

Chapter 8: User-Defined Windows — Describes user-defined windows and user-defined messages, gives examples of uses, and explains the application calls for these features.

Chapter 9: Internal Phrase Editing — Defines what a NaturalLink internal phrase is and how the internal phrases can be modified using the Phrase Editor utility.

NaturalLink is a trademark of Texas Instruments Incorporated.

Chapter 10: Window Manager Input Devices — Discusses the Window Manager keyboard and the functions invoked by it. Explains how to use the Function Key Change utility to assign Window Manager functions to specified keys. Also describes input to Window Manager from other input devices via an application Input routine.

Appendix A: Equipment Requirements — Lists the hardware and software required to use Window Manager.

Appendix B: Helpful Hints — Provides further explanations, examples, and hints on how to use Window Manager features.

Appendix C: Computer-Specific Information — Describes the computers supported by Window Manager, along with special considerations for each computer.

Appendix D: C Interface — Presents Lattice C Compiler™ coding translation tables, routines, and sample link streams.

Appendix E: Pascal Interface — Presents MS™-Pascal coding translation tables, routines, and sample link streams.

Appendix F: FORTRAN Interface — Presents MS-FORTRAN coding translation tables, routines, and sample link streams.

Appendix G: Compiled BASIC Interface — Presents coding translation tables, routines, and sample link streams to be used with the MS-BASIC Compiler.

Appendix H: Error Codes — Lists error code numbers and messages.

Lattice C Compiler is a trademark of Lattice Incorporated.

MS is a trademark of Microsoft Corporation.

The following documents contain information about the Texas Instruments Professional Computer, the NaturalLink software, and the MS-DOS operating system.

<i>Document Title</i>	<i>Part Number</i>
<i>Texas Instruments Professional Computer: Getting Started</i>	2223203-0001
<i>Texas Instruments NaturalLink Toolkit Reference Manual</i>	2240316-0001
<i>NaturalLink Technology Workbook</i>	2243736-0001
<i>Texas Instruments Professional Computer Operating Instructions</i>	2223116-0001
<i>Texas Instruments Professional Computer Technical Reference Manual</i>	2223216-0001
<i>MS-DOS Operating System</i>	2223133-0001

CONTENTS

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
1	Introduction	
1.1	Menu-Based Interactive Interfaces	1-3
1.1.1	Different Types of Menus	1-3
1.2	Window Manager	1-5
1.2.1	Windows and Items	1-5
1.2.2	Displays	1-6
1.2.3	Screens	1-6
1.3	Screen Builder	1-7
1.4	Message Builder	1-7
1.5	Message Manager	1-7
1.6	Control Flow	1-7
2	Window Manager Features	
2.1	Introduction	2-3
2.2	Window Types	2-3
2.2.1	List Window	2-3
2.2.2	Text Window	2-3
2.2.3	Display Window	2-3
2.2.4	Edit Window	2-3
2.2.5	File Window	2-3
2.2.6	User-Defined Window	2-4
2.3	Window Formats	2-4
2.4	Attributes	2-4
2.5	Controlled Cursor Movement	2-4
2.6	Search Feature	2-4
2.7	Scrolling	2-4
2.8	Help Key	2-5
2.9	Application-Assigned Function Keys	2-5
3	Window Attributes	
3.1	Attributes	3-3
3.2	Window Attributes With String Values	3-3
3.2.1	Window Label	3-3
3.2.2	File Window Pathname	3-4
3.3	Window Attributes With Numeric Values	3-4
3.3.1	Window Position	3-6
3.3.2	Window Format	3-6

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
3.3.2.1	Window Type.....	3-6
3.3.2.2	Number of Columns.....	3-7
3.3.2.3	Window Priority.....	3-8
3.3.2.4	Multiple-Selection Window.....	3-8
3.3.2.5	Pop-Up Window.....	3-9
3.3.2.6	Special Repaint on Receive.....	3-9
3.3.2.7	First Item to Be Displayed.....	3-9
3.3.2.8	Show Last Item When Painted.....	3-9
3.3.2.9	Center All Items.....	3-10
3.3.2.10	Don't Redisplay Current Items.....	3-10
3.3.2.11	Disable Multiple-Line Items.....	3-10
3.3.2.12	Allow Cursor to Enter Window.....	3-10
3.3.2.13	Multiple-Column Order.....	3-11
3.3.2.14	Maximum Item Label Length.....	3-11
3.3.2.15	Item Label Justification.....	3-11
3.3.2.16	Active Window.....	3-11
3.3.2.17	Number of Items in Window.....	3-11
3.3.3	Active Window Attributes.....	3-12
3.3.4	Inactive Window Attributes.....	3-12
3.3.5	Window Label Attributes.....	3-12
3.3.5.1	Invisible Label.....	3-12
3.3.5.2	Label Position.....	3-13
3.3.5.3	Centered Label.....	3-13
3.3.5.4	Blinking Label.....	3-13
3.3.5.5	Underlined Label.....	3-13
3.3.5.6	Reverse Video Label.....	3-13
3.3.5.7	Label Intensity/Color.....	3-13
3.3.5.8	Use Item Intensity.....	3-13
3.3.6	Cursor Attributes.....	3-14
3.3.6.1	Size and Type.....	3-14
3.3.6.2	Blinking Cursor.....	3-14
3.3.6.3	Underlined Cursor.....	3-14
3.3.6.4	Reverse Video Cursor.....	3-14
3.3.6.5	Cursor Intensity/Color.....	3-14
3.3.6.6	Use Item Intensity.....	3-14
3.3.6.7	Don't Delete Cursor.....	3-15
3.3.7	Border Attributes.....	3-15
3.3.7.1	Bordered Window.....	3-15
3.3.7.2	Border Takes up Space.....	3-15
3.3.7.3	Display Scroll Markers.....	3-15
3.3.7.4	Reverse Video Border.....	3-15
3.3.7.5	Border Intensity/Color.....	3-16
3.4	Item-Level Attributes.....	3-16
3.5	Item-Level Attributes With String Values.....	3-16
3.5.1	Item Label.....	3-16

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
3.5.2	Item Text	3-16
3.6	Item-Level Attributes With Numeric Values	3-16
3.6.1	Item-Format Attributes	3-17
3.6.1.1	Visible	3-17
3.6.1.2	Unselectable	3-17
3.6.1.3	Displayed	3-18
3.6.1.4	Maximum Edit Field Length	3-18
3.6.1.5	Edit Field Datatype	3-18
3.6.1.6	Required Edit Field	3-18
3.6.1.7	Echo Edit Field Input	3-18
3.6.1.8	Chosen/Enable Attribute	3-19
3.6.2	Item-Chosen Attributes	3-19
3.6.2.1	Blinking Chosen Item	3-19
3.6.2.2	Underlined Chosen Item	3-19
3.6.2.3	Reverse Video Chosen Item	3-19
3.6.2.4	Chosen Item Intensity/Color	3-19
3.6.3	Item Label Attributes	3-19
3.6.3.1	Blinking Label	3-19
3.6.3.2	Underlined Label	3-19
3.6.3.3	Reverse Video Label	3-19
3.6.3.4	Label Intensity/Color	3-19
3.6.3.5	Use Item Intensity	3-19

4

Screen Builder

4.1	Using Screen Builder	4-3
4.2	Screen-Level Commands	4-4
4.2.1	Edit Window	4-4
4.2.2	Add Window	4-5
4.2.3	Copy Window	4-5
4.2.4	Insert Window	4-5
4.2.5	Delete Window	4-5
4.2.6	Change Help File	4-5
4.2.7	Draw Screen	4-6
4.2.8	Test Screen	4-7
4.2.9	Load Screen	4-7
4.2.10	Save Screen	4-8
4.2.11	List Screen Attributes	4-8
4.2.12	Exit	4-8
4.3	Window-Level Commands	4-9
4.3.1	Set Window Position	4-10
4.3.2	Set Window Format	4-10
4.3.3	Set Inactive Window Attributes	4-10
4.3.4	Set Active Window Attributes	4-10
4.3.5	Edit Window Label	4-11

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
4.3.6	Set Window Label Attributes	4-12
4.3.7	Set Cursor Attributes	4-12
4.3.8	Set Border Attributes	4-12
4.3.9	Attach Help to Window	4-12
4.3.10	Edit Item Table	4-13
4.3.11	Draw Window	4-13
4.3.12	Test Window	4-13
4.4	Item-Level Commands	4-13
4.4.1	Edit Item Text	4-14
4.4.2	Set Item Attributes	4-15
4.4.3	Edit Item Label	4-15
4.4.4	Set Item Label Attributes	4-16
4.4.5	Add, Copy, Insert, and Delete Item	4-16
4.4.6	Attach Help to Item	4-16
4.5	The Help File	4-16
4.5.1	Changing the Help File	4-16
4.5.2	Setting a Path for the Help File	4-17
4.6	Attaching Help to Windows and Items	4-18
4.6.1	Add Help Message to List	4-19
4.6.2	Insert Help Message Into List	4-20
4.6.3	Delete Help Message From List	4-20
4.6.4	View Help Message	4-20
4.6.5	Specify Help Message	4-20

5

Message Builder

5.1	Using Message Builder	5-3
5.2	Message Builder Options	5-3
5.2.1	Add Message	5-5
5.2.2	Modify Message	5-6
5.2.3	Rename Message	5-6
5.2.4	Change Message Type/Class	5-7
5.2.5	Copy Message	5-7
5.2.6	Reuse Message	5-7
5.2.7	Specify Window Coordinates	5-7
5.2.8	Delete Message	5-8
5.2.9	View Message	5-8
5.2.10	List Messages	5-8
5.2.11	Quit	5-9
5.3	Default Message	5-9
5.4	Using Message Manager	5-9
5.4.1	Variable Text	5-10
5.4.2	Message Manager Errors	5-11

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
6	Window Manager Callable Routines	
6.1	Procedure Calls.....	6-3
6.1.1	Initialize Window Manager.....	6-3
6.1.2	Load Screen File.....	6-4
6.1.3	Add Window.....	6-4
6.1.4	Select Window.....	6-4
6.1.5	Refresh Window.....	6-5
6.1.6	Display Window.....	6-5
6.1.7	Receive From Window.....	6-5
6.1.8	Release Window.....	6-6
6.1.9	Delete Window.....	6-7
6.1.10	Save Screen.....	6-7
6.1.11	Unload Screen.....	6-7
6.1.12	Reset Window Manager.....	6-7
6.1.13	Add Item.....	6-7
6.1.14	Insert Item.....	6-8
6.1.15	Delete Item.....	6-8
6.1.16	Create Item Table.....	6-8
6.1.17	Create Window.....	6-9
6.1.18	Get Attribute Value.....	6-9
6.1.19	Get String Value.....	6-9
6.1.20	Set Attribute Value.....	6-9
6.1.21	Set String Value.....	6-9
6.1.22	Clear Screen.....	6-9
6.1.23	Display Message From Message Manager.....	6-10
6.1.24	Reset NaturalLink Memory.....	6-10
6.2	Typical Calling Sequence.....	6-11
7	Application Validation Routine	
7.1	Definition.....	7-3
7.2	Uses for the Routine.....	7-3
7.3	When Validation Is Done.....	7-4
7.4	How It Works.....	7-4
7.5	Parameters and Return Status.....	7-5
7.5.1	Datatype Parameter.....	7-5
7.5.2	Return Status for Validation Process.....	7-5
7.5.3	Item Number Parameter.....	7-6
7.5.4	Typical Algorithm for Validation Code.....	7-7
7.6	Restrictions on Use of Validation Routine.....	7-8
7.7	Dummy Validation Routine.....	7-8

Paragraph	Title	Page
8	User-Defined Windows	
8.1	Introduction	8-3
8.2	Functionality	8-3
8.2.1	Specifying a UDW	8-3
8.2.2	How UDWs Work With Window Manager Run Time	8-4
8.2.2.1	Receive Call.....	8-4
8.2.2.2	Display Call	8-6
8.2.2.3	Delete Call.....	8-6
8.2.2.4	Other WM calls.....	8-7
8.3	User-Defined Messages	8-7
8.3.1	Specifying the Message	8-7
8.3.2	How the UDM Works	8-7
8.4	Uses for UDWs/UDMs.....	8-8
8.4.1	Graphic Title Window	8-8
8.4.2	Command Window	8-8
8.4.3	Selection From Icons.....	8-8
8.4.4	Graphical Help Messages	8-8
8.5	UDW/UDM Application Calls	8-9
8.5.1	Application Receive Call.....	8-9
8.5.2	Application Display Call.....	8-10
8.5.3	Application Delete Call	8-10
8.5.4	Application Message Manager Call.....	8-11
8.6	Making Window Manager Calls Inside a UDW Routine	8-11
9	Internal Phrase Editing	
9.1	Introduction	9-3
9.2	Examples of Internal Phrases.....	9-3
9.3	Modifying Internal Phrases	9-4
9.3.1	Modifying Phrases Before Linking the Application	9-4
9.3.2	Modifying Phrases Using a Phrase Input File (Phrase Editor Utility).....	9-4
9.4	Using PBUILD	9-5
9.4.1	Phrase Editor Commands	9-5
9.4.1.1	Edit a Phrase	9-6
9.4.1.2	Restore Default Phrases.....	9-6
9.4.1.3	Print Phrases to File	9-6
9.4.1.4	Save Phrases to File	9-7
9.4.1.5	Exit	9-7
9.5	Usage of the NLXPHRAS.NM\$ File	9-7
9.6	Listing of Internal Phrases.....	9-7

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
10	Window Manager Input Devices	
10.1	Overview	10-3
10.2	Window Manager Keyboard Input	10-3
10.2.1	Window Manager Keyboard Input Values	10-4
10.3	Window Manager Input Default Functions.....	10-4
10.3.1	Select — Default: ENTER Key	10-7
10.3.2	Proceed — Default: F10 Key	10-7
10.3.3	Page Scroll — Default: F1 and F2 Keys.....	10-7
10.3.4	Help — Default: F7 Key.....	10-8
10.3.5	Backup — Default: F8 Key	10-8
10.3.6	Next Item/Line Scrolling — Default: Arrow Keys	10-8
10.3.7	Move Right/Left One Item — Default: TAB/SHIFT-TAB key	10-9
10.3.8	Top/Bottom — Default: HOME Key	10-9
10.3.9	Next Window — Default: CTRL-L/R Arrow Keys and CTRL-PgUp/PgDn Keys	10-9
10.3.10	Next Active Window — Default: CTRL-HOME Key.....	10-9
10.3.11	Move to Start/End of Line — Default: CTRL-F1/F2 Keys	10-9
10.3.12	Move Left or Right One Word — Default: CTRL-F3/F4 Keys	10-9
10.3.13	Insert — Default: INS Key.....	10-10
10.3.14	Delete — Default: DEL Key	10-10
10.3.15	Delete From the Cursor Out — Default: CTRL-F5 Key.....	10-10
10.3.16	Backspace — Default: BACKSPACE Key.....	10-10
10.3.17	Printable Keys — Default: Keys With Key Codes Between 00H and FFH	10-10
10.3.18	Using Printable Keys in the Search Mode.....	10-11
10.3.19	Input Unknown to Window Manager	10-11
10.4	Changing Function Key Assignments.....	10-11
10.5	Application Input Routine	10-13
10.5.1	Definition and Use.....	10-14
10.5.2	APPINP Return Code	10-14
10.5.3	Dummy APPINP Routine	10-14

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
A	Equipment Requirements	
A.1	Equipment Requirements.....	A-3
A.1.1	Equipment Necessary for Interface Development	A-3
A.1.2	Equipment Necessary for User Operation	A-4
B	Helpful Hints	
B.1	Introduction	B-3
B.2	Window Types	B-3
B.2.1	List Windows.....	B-3
B.2.2	Text Windows.....	B-4
B.2.3	Display Windows	B-4
B.2.4	Edit Windows.....	B-4
B.2.4.1	Maximum Edit Field Length.....	B-4
B.2.4.2	Required Edit Field.....	B-5
B.2.4.3	Echo Edit Field Input	B-5
B.2.4.4	Multiple Selection in an Edit Window	B-5
B.2.5	File Windows	B-5
B.3	Window Labels.....	B-6
B.3.1	Top Window Label	B-6
B.3.2	Bottom Window Label.....	B-7
B.3.3	Left Window Label.....	B-7
B.3.4	Right Window Label	B-7
B.4	Window Formats.....	B-8
B.4.1	Single-Column Format.....	B-9
B.4.2	Multiple-Column Format	B-9
B.4.3	Free Format	B-10
B.5	Item Labels.....	B-11
B.5.1	Maximum Item Label Length	B-12
B.5.2	Left-Justified Labels	B-12
B.5.3	Right-Justified Labels.....	B-12
B.5.4	Free-Format Labels	B-12
B.6	Special Window Manager Features	B-13
B.6.1	Use of Borderless Windows and Multiple Window Effects	B-13
B.6.1.1	Adding Multiple Windows in Order	B-14
B.6.1.2	Reasons for Using Multiple Windows.....	B-14
B.6.2	Use of Display Windows	B-15
B.6.3	Creating and Using Blank Lines	B-16
B.6.4	Unselectable Items.....	B-16
B.6.5	Use of Invisible Items.....	B-17
B.6.6	When to Use Multiple-Selection Windows	B-17
B.6.7	Simulating a Multiple-Selection Window	B-18
B.6.7.1	Using the Chosen/Enable Item Attribute	B-18
B.6.7.2	Simulating Cursor Movement	B-19

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
B.6.7.3	Using the Don't Redisplay Current Items Attribute.....	B-19
B.6.7.4	Using the Displayed Attribute.....	B-20
B.6.7.5	First Item to Be Displayed.....	B-20
B.6.8	Using Windows for More Than One Purpose.....	B-21
B.6.9	Special Repaint on Receive.....	B-21
B.7	Designing Screen Files.....	B-22

C

Computer-Specific Information

C.1	Introduction.....	C-3
C.2	Computers Supported.....	C-3
C.2.1	List of Computers.....	C-3
C.2.2	Machine Detection.....	C-3
C.3	TI Computers.....	C-4
C.3.1	TI BUSINESS-PRO™ Computer.....	C-4
C.3.2	TIPC/TIPPC Function Key Differences.....	C-5
C.3.3	TIPC Character Codes.....	C-5
C.3.4	TI PRO-LITE™ Computer.....	C-9
C.3.4.1	Considerations for TI PRO-LITE Computer.....	C-11
C.3.4.2	LCD Flag.....	C-12
C.4	IBM® PC, PC/XT™, and Personal Computer AT™.....	C-12
C.4.1	Underlining.....	C-12
C.4.2	Reverse Video.....	C-13
C.4.3	Intensity/Color.....	C-13
C.4.4	Cursor Attributes.....	C-13
C.4.5	Key Codes.....	C-13

D

C Interface

D.1	Introduction.....	D-3
D.2	Parameter Characteristics.....	D-3
D.2.1	Zero-Base Values.....	D-3
D.2.2	Integers.....	D-3
D.2.3	Character Strings.....	D-3
D.2.3.1	Passing Strings to Window Manager.....	D-4
D.2.3.2	Strings Returned by Window Manager.....	D-4
D.3	Callable Routines.....	D-5
D.3.1	Function Return Codes.....	D-5
D.4	Routines Called by Window Manager.....	D-14
D.5	C Compiling Requirements.....	D-17
D.5.1	Include File.....	D-17
D.6	Memory Considerations.....	D-19
D.7	Linking Considerations.....	D-20

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
D.7.1	Memory Model Differences.....	D-21
D.7.2	Dummy Routines Library.....	D-21
D.7.3	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files	D-21
D.7.4	Object Code Segments	D-22
D.8	Restrictions	D-22

E

Pascal Interface

E.1	Introduction.....	E-3
E.2	Parameter Characteristics	E-3
E.2.1	Zero-Base Values	E-3
E.2.2	Integers	E-3
E.2.3	Character Strings	E-3
E.2.3.1	Passing Strings to Window Manager.....	E-4
E.2.3.2	Strings Returned by Window Manager.....	E-4
E.3	Callable Routines	E-5
E.3.1	Function Return Codes	E-5
E.4	Routines Called by Window Manager	E-14
E.5	Pascal Compiling Requirements	E-17
E.5.1	Include Files	E-17
E.6	Memory Considerations.....	E-19
E.7	Linking Considerations	E-20
E.7.1	Dummy Routines Library	E-20
E.7.2	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files.....	E-21
E.7.3	Object Code Segments	E-21
E.8	Restrictions.....	E-21

F

FORTRAN Interface

F.1	Introduction.....	F-3
F.2	Parameter Characteristics	F-3
F.2.1	One-Base Values	F-3
F.2.2	Integers.....	F-3
F.2.3	Character Strings	F-3
F.2.3.1	Passing Strings to Window Manager	F-4
F.2.3.2	Strings Returned by Window Manager	F-4
F.3	Callable Routines.....	F-5
F.3.1	Function Return Codes	F-5
F.4	Routines Called by Window Manager	F-14
F.5	FORTRAN Compiling Requirements	F-17
F.5.1	Common Blocks and Include Files	F-17
F.5.2	Initialization of Window Manager	F-17
F.6	Memory Considerations	F-20
F.7	Linking Considerations	F-21
F.7.1	Dummy Routines Library	F-21
F.7.2	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files	F-21
F.7.3	Object Code Segments	F-21
F.8	Restrictions.....	F-22

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
G	Compiled BASIC Interface	
G.1	Introduction	G-3
G.2	Parameter Characteristics	G-3
G.2.1	Zero-Base Values	G-3
G.2.2	Names of Variables	G-3
G.2.3	Integers	G-3
G.2.4	Character Strings	G-3
G.2.4.1	Passing Strings to Window Manager	G-4
G.2.4.2	Strings Returned by Window Manager	G-4
G.3	Callable Routines	G-5
G.3.1	Procedure Return Codes	G-5
G.4	Routines Called by Window Manager	G-15
G.5	BASIC Compiling Requirements	G-22
G.5.1	Include Files	G-22
G.6	Memory Considerations	G-25
G.7	Linking Considerations	G-26
G.7.1	Dummy Routines Library	G-26
G.7.2	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files	G-26
G.7.3	Object Code Segments	G-26
G.8	Restrictions	G-26

H **Window Manager Error Codes**

Index

<i>Figure</i>	<i>Title</i>	<i>Page</i>
Figures		
1-1	Example Menu — Window Manager Reference Manual Topics	1-4
1-2	Example Window	1-5
1-3	Screen and Message Definition	1-8
1-4	Interactive Screen Management	1-9
4-1	Screen Builder Main Menu — Screen Commands	4-4
4-2	Window Level Menu	4-9
4-3	Item-Level Menu	4-14
4-4	Attach Help Messages Menu	4-19
5-1	Message Builder Utility	5-4
9-1	Internal Error Message	9-3
9-2	Help Message	9-4
9-3	Phrase Editor Commands Screen	9-5
9-4	Enter Phrase Pop-Up Window	9-6
10-1	KBUILD Utility Menu	10-12
B-1	Window Label Positions	B-6
B-2	Window Column Formats	B-8
B-3	Item Labels	B-11
B-4	Window Label Attributes	B-13
B-5	Invisible Borders	B-14
C-1	TI BUSINESS-PRO™ Professional Computer U.S. Keyboard	C-4
C-2	TIPC U.S. Keyboard Layout and Scan Codes	C-9
C-3	TI PRO-LITE™ Computer U.S. Keyboard	C-10
C-4	IBM® PC and PC/XT™ U.S. Keyboard	C-14
C-5	IBM Personal Computer AT™ U.S. Keyboard	C-14

	<i>Table Title</i>	<i>Page</i>
Tables	3-1 Window-Level Attributes	3-6
	3-2 Item-Level Attributes	3-17
	9-1 Window Manager Internal Phrases That Can Be Modified	9-8
	10-1 Default Keys in Inactive Windows	10-4
	10-2 Default Keys in Active Windows	10-6
	C-1 TI Computer Function Key Differences	C-5
	C-2 Standard U.S. Keyboard Character Codes	C-7
	C-3 Key Codes for the IBM Personal Computer	C-16
	D-1 C Interface Routines	D-6
	D-2 Application-Provided Routines	D-14
	D-3 C Data Declarations for Window Attributes	D-17
	E-1 Pascal Interface Routines	E-6
	E-2 Application-Provided Routines	E-14
	E-3 Pascal Data Declarations for Window Attributes	E-17
	F-1 FORTRAN Interface Routines	F-6
	F-2 Application-Provided Routines	F-14
	F-3 Data Declarations for the FORTRAN Interface	F-18
	G-1 Compiled BASIC Interface Routines	G-6
	G-2 Application-Provided Routines	G-19
	G-3 Data Declarations for the Compiled BASIC Interface	G-23

INTRODUCTION

1

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
1.1	Menu-Based Interactive Interfaces	1-3
1.1.1	Different Types of Menus	1-3
1.2	Window Manager	1-5
1.2.1	Windows and Items	1-5
1.2.2	Displays	1-6
1.2.3	Screens	1-6
1.3	Screen Builder	1-7
1.4	Message Builder	1-7
1.5	Message Manager	1-7
1.6	Control Flow	1-7

Menu-Based Interactive Interfaces

1.1 NaturalLink Window Manager and its associated utilities, NaturalLink Screen Builder and NaturalLink Message Builder, are software tools that can be used to create and control menu-based interfaces to application programs in an interactive computing environment. Although the use of menus is common in interface design, the Window Manager and Screen Builder utilities simplify both the creation and use of menus.

Different Types of Menus

1.1.1 A variety of menus can be produced with these NaturalLink software tools. The simplest menu presents a list of options to the user from which items can be selected by placing the cursor on the choice desired and pressing the **ENTER** key. Window Manager passes the selection to the application program, and the application program performs the action requested.

More sophisticated NaturalLink menus can request a user to type specific data or to construct a command sentence using nouns, verbs, objects, or other sentence elements selected from interactive menu lists. Areas in the menu can be used to display information for improved user understanding. The application designer can add a Help message to the menu and Window Manager will display that message at the touch of a key. In addition, Window Manager can display special messages, designed with the aid of Message Builder.

Window Manager is also flexible in the way it can present menus. The full use of color and special attributes, such as blinking, underlining, and reverse video, are available.

The menu in Figure 1-1 illustrates how you could use Window Manager and Screen Builder to present the topics in this manual.

Figure 1-1

Example Menu — Window Manager Reference Manual Topics

WINDOW MANAGER REFERENCE MANUAL	
Chapter	Topic
1	Introduction
2	Window Manager Features
3	Window Attributes
4	Screen Builder
5	Message Builder
6	Window Manager Callable Routines
7	Window Manager Validation Routine
8	User Defined Windows
9	Internal Phrase Editing
10	Window Manager Input Devices

Put the cursor on the desired topic and press the ENTER key to select it.

You can design, view, and test this menu using Screen Builder. Window Manager, in turn, controls the interaction between the user and the application program for which the menu was created. This includes cursor movement, scrolling of the window contents, topic selection, and displaying of messages built with the aid of Message Builder. The menu can easily be modified to achieve the results desired for a specific interactive environment. In short, Window Manager, Screen Builder, and Message Builder are versatile tools for producing NaturalLink Interfaces. The rest of this chapter describes the specific elements of these tools in greater detail.

Window Manager

1.2 NaturalLink Window Manager is a screen management system that controls the interaction between an application program and a user. Window Manager accepts input from the user through windows, window displays, and screens, and either sends data to the application program for further processing or displays a response to the user.

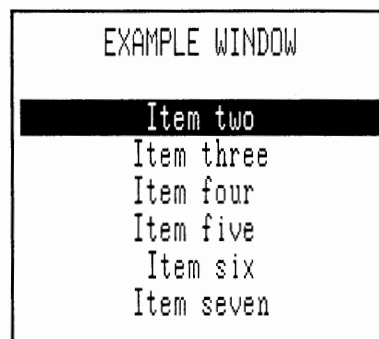
Windows and Items

1.2.1 The window, a rectangular area displayed on a video device, is the principal unit controlled by Window Manager. The characteristics of a window can vary considerably. A window can be either bordered or borderless. It can contain text and/or graphics, or it can be completely empty. Each part of a window can be displayed in a different intensity/color. A single window can fill the entire video display or only part of it. The display can also contain a number of windows, each a different size and each having different attributes (attributes are discussed in detail in Chapter 3, Window Attributes). The number of windows displayed is governed by the needs of the application program to which Window Manager provides the high-level interface.

A window can contain a label and one or more window items. Window labels are frequently used for window identification, while window items are the words and phrases used to communicate with the user and the application program. For example, Figure 1-2 illustrates a basic window.

Figure 1-2

Example Window



EXAMPLE WINDOW is the window label, and the phrases (Item one, Item two, and so on) are the actual window items. Window items can be selected, edited, or scrolled, depending on the characteristics of the window. Window Manager contains logic necessary to execute the following functions:

- Display windows on a video device
- Move a cursor from item to item within a window and from one window to another
- Scroll the window contents
- Select a window item
- Modify the window contents
- Redraw a window
- Delete a window

In order to perform these functions, Window Manager must know the characteristics of a window. Window characteristics are recorded in a data structure called the window description (also referred to as the window definition), which can be created and recorded in a file prior to application program execution using Window Manager. Window descriptions are specified when the windows are initially defined with Screen Builder and can be modified by the interaction between application software and Window Manager during program execution.

Displays 1.2.2 A display is a set of windows simultaneously displayed on a video device. Although a display can consist of more than one window, input can be processed from only one window at a time. That window must be active before Window Manager permits a user to select data.

Screens 1.2.3 A screen is a collection of logically related windows. A window normally belongs to a screen if there is a possibility that the window will need to be displayed during the course of an interactive session. It is not necessary that all windows of a screen be displayed simultaneously. The number of windows in a logically related collection varies depending upon the needs of the application.

Screen Builder

1.3 Screen Builder is an interactive utility that facilitates development of a window description. This utility elicits information about each window, such as its size and location, and stores this descriptive information in a file for use by Window Manager during execution of the application software. Screen Builder can also be used to modify window descriptions as required in the course of developing or enhancing a user interface.

Message Builder

1.4 Message Builder is an interactive utility that aids the application designer in constructing messages for use in an application program. Four types of messages can be constructed:

- Help
- Error
- Warning
- Please Note

Each message can be classified as either a text message or a user-defined message. Once constructed, these messages are stored in a file and can be selectively displayed by Message Manager.

Message Manager

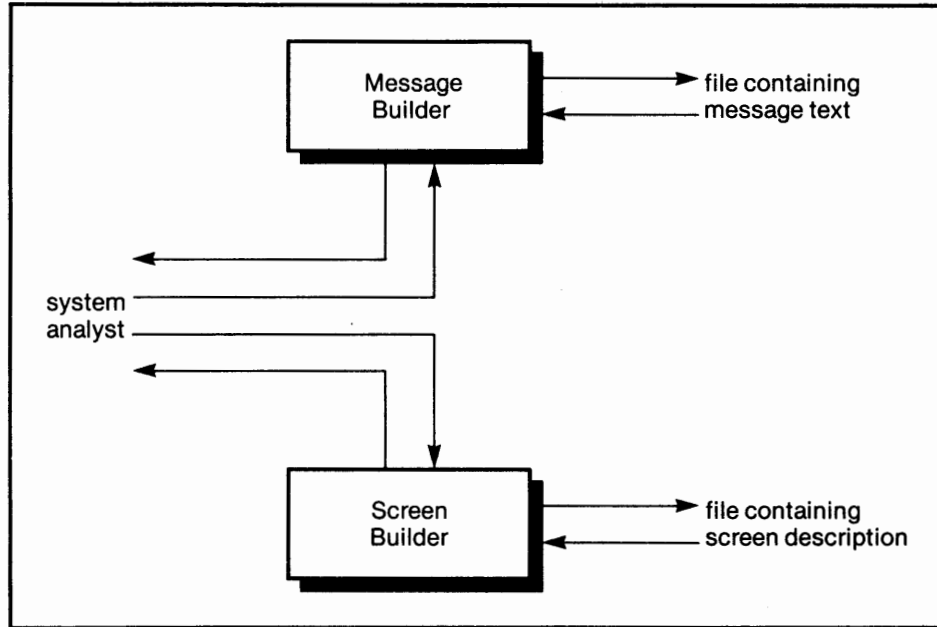
1.5 Message Manager is a utility that displays the messages constructed with the aid of Message Builder. Message Manager reads the binary file created by Message Builder and displays a requested message inside a window. If coordinates are specified for a message, those coordinates are used to position the window on the screen. If coordinates are not specified, Message Manager shrinks the window to fit the message and places the window in the center of the screen.

Control Flow

1.6 Figures 1-3 and 1-4 illustrate the major processes and data structures involved in developing and using an interactive interface. As shown in Figure 1-3, an application developer uses Screen Builder to create a screen description and Message Builder to create a set of messages. The screen descriptions and message text are stored in separate files for use during execution of an application program.

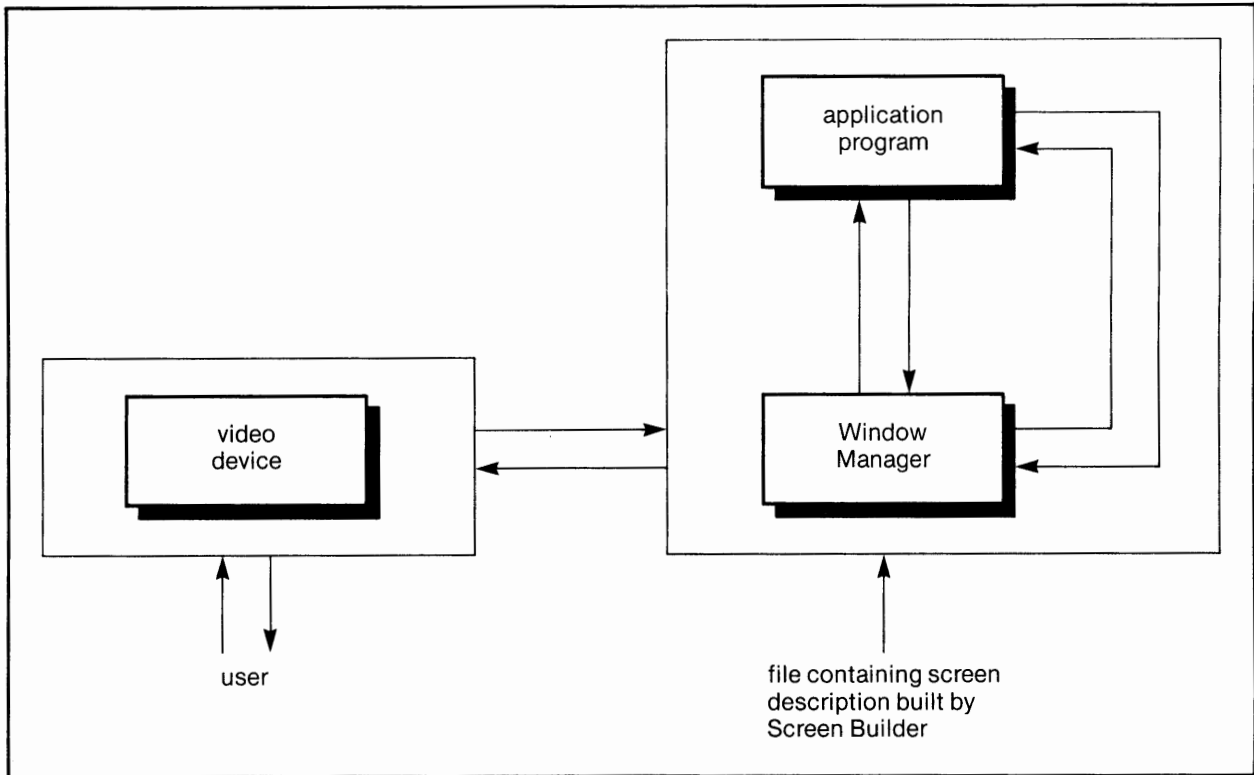
Figure 1-3

Screen and Message Definition



As illustrated in Figure 1-4, an application software program can call upon Window Manager to handle interactive dialogue with the user. In doing this, Window Manager reads window descriptions from a file and displays the windows on the video device in accordance with the procedure calls issued by the application program. Both Window Manager and the application program can call on Message Manager to display message text.

Figure 1-4 Interactive Screen Management



In a typical application, the calls proceed as follows:

1. Load screen description from a file into memory.
2. Display selected windows on the video device.
3. Activate one or more of the windows.
4. Receive input from an active window.
5. Display a result in a window.
6. Repeat, if desired, from Step 2.

WINDOW MANAGER FEATURES

2

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
2.1	Introduction.....	2-3
2.2	Window Types.....	2-3
2.2.1	List Window.....	2-3
2.2.2	Text Window.....	2-3
2.2.3	Display Window.....	2-3
2.2.4	Edit Window.....	2-3
2.2.5	File Window.....	2-3
2.2.6	User-Defined Window.....	2-4
2.3	Window Formats.....	2-4
2.4	Attributes.....	2-4
2.5	Controlled Cursor Movement.....	2-4
2.6	Search Feature.....	2-4
2.7	Scrolling.....	2-4
2.8	Help Key.....	2-5
2.9	Application-Assigned Function Keys.....	2-5

Introduction

2.1 Several features make Window Manager a powerful software development tool. These include the availability of six window types, two categories of window formats, numerous window and item attributes, full cursor control, built-in help capability, and application-assigned function keys. This chapter provides an overview of these features.

Window Types

2.2 Window Manager performs certain functions on the basis of window type. The type of window in which an item is displayed determines whether that item can be edited, selected, scrolled, or displayed.

List Window **2.2.1** The list window is the basic window type. It contains a list of items in various formats through which the cursor can be moved and from which items can be selected.

Text Window **2.2.2** A text window has the same appearance as a list window. Items in a text window can only be scrolled, however. The cursor is not visible in a text window and text selection is not permitted. Text can be scrolled either one line at a time or a window at a time.

Display Window **2.2.3** As its name implies, a display window is used for display purposes only. The essential difference between a display window and other types of windows is that the cursor can never enter a display window. A display window provides information for the user while preventing manipulation of window contents.

Edit Window **2.2.4** An edit window contains items having text that can be edited. The user can interactively modify, delete, or add to the item text in an edit window. In addition, the application designer can specify one or more required fields (alphanumeric fields in which the user is required to enter data before a procedure can be completed) in an edit window. Data entered in these edit fields can be either echoed to the screen or suppressed from view for data security.

File Window **2.2.5** A file window differs from a text window in that it can be used to display text files not associated with Window Manager. When a file pathname is supplied, Window Manager makes the necessary format adjustments to display the file in a window. Text files displayed in this manner can be scrolled for viewing, but are not available for selection.

User-Defined Window **2.2.6** User-defined windows (UDWs) provide unlimited flexibility in Window Manager. They display information in ways that Window Manager does not directly provide. This includes the display of graphical pictures and the display and selection of graphical text and icons. Control of UDWs is shared jointly by Window Manager and the application. For more explanation, see Chapter 8, User-Defined Windows.

Window Formats **2.3** Window Manager supports two categories of window formats:

- Columnar format — Single or multiple columns spaced evenly within the window.
- Free format — Items displayed one after another with one space between each item. Window formats are discussed further in Chapter 3, Window Attributes.

Attributes **2.4** Window Manager enables you to set attributes at different levels. Basic attributes include blinking, underlining, reverse video, and screen intensity. The specific screen attributes supported may vary, depending on the hardware. For specific attributes supported on different hardware, see Appendix C, Computer-Specific Information. Additional attributes include centering, justification, visibility, and selectability. Attributes can be set for active and inactive windows, window labels, window border, item labels, items, and the cursor.

Controlled Cursor Movement **2.5** Window Manager controls cursor movement. The cursor can be moved vertically and horizontally from item to item, window to window, or active window to active window. Faster cursor movement is provided by application-assigned keys which move the cursor to the top or bottom item in a window or scroll through items a window length at a time.

Search Feature **2.6** A search feature is available for rapid movement of the cursor to a specific item in a list window. The search capability is invoked by entering the beginning character(s) of the desired item. When any displayable character is pressed, the cursor is placed on the first item that begins with that character. The search can go further by typing additional characters. Window Manager remains in the search mode until an unprintable key, such as an **arrow** key or the **HOME** key, is pressed.

Scrolling **2.7** Windows can be scrolled either one item (line scrolling) or one window (page scrolling) at a time. Automatic scrolling occurs when the cursor reaches the top or bottom of a window and more items remain than can be displayed at one time.

Help Key

2.8 A Help key can be designated to provide the user with further information about windows or window items. Window Manager performs all processing of the designated Help key. Help messages are stored in separate files, and any number of Help messages can be chained to an item or window. When the user presses the Help key, Window Manager displays the first Help message chained to the item or window. If the Help key is pressed again while a Help message is currently displayed, the next Help message appears. The user can back out of Help messages one step at a time or cancel the Help messages immediately with the aid of application-assigned keys.

Application- Assigned Function Keys

2.9 Window Manager supports a prescribed set of operations that can be invoked with function keys. However, you can assign the key to be used for a particular function. For example, if the application software does not permit the use of **F1** and **F2** keys as window scrolling keys, you can assign other keys instead.

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
3.1	Attributes.....	3-3
3.2	Window Attributes With String Values.....	3-3
3.2.1	Window Label	3-3
3.2.2	File Window Pathname.....	3-4
3.3	Window Attributes With Numeric Values	3-4
3.3.1	Window Position	3-6
3.3.2	Window Format	3-6
3.3.2.1	Window Type.....	3-6
3.3.2.2	Number of Columns	3-7
3.3.2.3	Window Priority.....	3-8
3.3.2.4	Multiple-Selection Window	3-8
3.3.2.5	Pop-Up Window	3-9
3.3.2.6	Special Repaint on Receive	3-9
3.3.2.7	First Item to Be Displayed	3-9
3.3.2.8	Show Last Item When Painted.....	3-9
3.3.2.9	Center All Items	3-10
3.3.2.10	Don't Redisplay Current Items	3-10
3.3.2.11	Disable Multiple-Line Items	3-10
3.3.2.12	Allow Cursor to Enter Window	3-10
3.3.2.13	Multiple-Column Order	3-11
3.3.2.14	Maximum Item Label Length	3-11
3.3.2.15	Item Label Justification.....	3-11
3.3.2.16	Active Window	3-11
3.3.2.17	Number of Items in Window	3-11
3.3.3	Active Window Attributes	3-12
3.3.4	Inactive Window Attributes.....	3-12
3.3.5	Window Label Attributes.....	3-12
3.3.5.1	Invisible Label	3-12
3.3.5.2	Label Position.....	3-13
3.3.5.3	Centered Label.....	3-13
3.3.5.4	Blinking Label	3-13
3.3.5.5	Underlined Label	3-13
3.3.5.6	Reverse Video Label	3-13
3.3.5.7	Label Intensity/Color	3-13
3.3.5.8	Use Item Intensity	3-13
3.3.6	Cursor Attributes.....	3-14
3.3.6.1	Size and Type.....	3-14

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
3.3.6.2	Blinking Cursor	3-14
3.3.6.3	Underlined Cursor	3-14
3.3.6.4	Reverse Video Cursor	3-14
3.3.6.5	Cursor Intensity/Color	3-14
3.3.6.6	Use Item Intensity	3-14
3.3.6.7	Don't Delete Cursor	3-15
3.3.7	Border Attributes	3-15
3.3.7.1	Bordered Window.....	3-15
3.3.7.2	Border Takes up Space	3-15
3.3.7.3	Display Scroll Markers.....	3-15
3.3.7.4	Reverse Video Border.....	3-15
3.3.7.5	Border Intensity/Color	3-16
3.4	Item-Level Attributes	3-16
3.5	Item-Level Attributes With String Values	3-16
3.5.1	Item Label.....	3-16
3.5.2	Item Text	3-16
3.6	Item-Level Attributes With Numeric Values.....	3-16
3.6.1	Item-Format Attributes	3-17
3.6.1.1	Visible.....	3-17
3.6.1.2	Unselectable.....	3-17
3.6.1.3	Displayed.....	3-18
3.6.1.4	Maximum Edit Field Length	3-18
3.6.1.5	Edit Field Datatype	3-18
3.6.1.6	Required Edit Field	3-18
3.6.1.7	Echo Edit Field Input.....	3-18
3.6.1.8	Chosen/Enable Attribute	3-19
3.6.2	Item-Chosen Attributes	3-19
3.6.2.1	Blinking Chosen Item.....	3-19
3.6.2.2	Underlined Chosen Item.....	3-19
3.6.2.3	Reverse Video Chosen Item	3-19
3.6.2.4	Chosen Item Intensity/Color	3-19
3.6.3	Item Label Attributes	3-19
3.6.3.1	Blinking Label	3-19
3.6.3.2	Underlined Label	3-19
3.6.3.3	Reverse Video Label.....	3-19
3.6.3.4	Label Intensity/Color	3-19
3.6.3.5	Use Item Intensity	3-19

Attributes

3.1 Attributes are properties assigned to windows and components of windows. Attributes include such characteristics as blinking, bordering, intensity/color, and reverse video. They govern how Window Manager displays and controls windows. Attribute values (character strings or numeric strings) are specified either with the aid of the Screen Builder utility during window construction or by direct command calls in an application program. The collection of window attributes is termed a window description. A collection of window descriptions constitutes a screen description. Several attribute values can be set at the window level; that is, they can be set at the level that defines a particular window. These window-level attributes can have character string values as well as numeric values. This chapter describes these attributes and their associated alphanumeric values.

In addition to this chapter, it is recommended that you read Appendix B, Helpful Hints. It provides more information on the function of window attributes in an interface screen and gives several examples.

Window Attributes With String Values

3.2 String value attributes at the window level can be used to label a window, provide instructions to the user, identify window contents, and so forth. The following attributes are specified using character string values.

Window Label

3.2.1 The window label is a word, line, or multiple lines of text placed at the top, bottom, left, or right side of a window. It is stationary in a window during all cursor movements; that is, it cannot be scrolled out of the window.

Window labels can consist of up to 25 lines, with each line containing a maximum of 80 characters. Any label line wider than the boundaries of the window will be truncated. If there are too many lines to fit into the window, only the number of lines that fit will be displayed.

The window label is stored as one continuous string, even if it occupies multiple lines. Each line of the label is separated by a line feed character (hexadecimal 0A). When you specify the window label in Screen Builder, these line feed characters are inserted for you. However, if you set the window label in your application program, ensure that your string has the line feed characters in the proper position to obtain the desired number of lines.

File Window Pathname

3.2.2 The file window pathname indicates to Window Manager the pathname of the file which is to be displayed in a file window. The file window pathname must be set to a text file pathname compatible with the operating system in use.

Window Attributes With Numeric Values

3.3 Many window-level attributes are set using numeric values. Table 3-1 provides a list of these attributes and their value ranges. The paragraphs following the table contain explanations of each.

Table 3-1 Window-Level Attributes

<i>Window Attributes</i>	<i>Value</i>	<i>Description</i>
Window Position:		
Left-Most Column	0 - 78	
Top Row	0 - 23	
Right-Most Column	1 - 79	
Bottom Row	1 - 24	
Window Format:		
Window Type	0 - 4, 10 - 99	0 = list, 1 = text, 2 = edit, 3 = file, 4 = display 10-99 = user-defined window
Number of Columns	0 - 80	0 = free format
Window Priority	0 - 16	defined by designer
Multiple Selection Window	0 or 1	0 = no, 1 = yes
Pop-up Window	0 or 1	0 = no, 1 = yes
Special Repaint on Receive	0 or 1	0 = no, 1 = yes
First Item to be Displayed	0 - N	Where N = one less than the number of items. First item displayed corresponds to value of N.
Show Last Item When Painted	0 or 1	0 = no, 1 = yes
Center All Items	0 or 1	0 = no, 1 = yes
Don't Redisplay Current Items	0 or 1	0 = no, 1 = yes
Disable Multiple Line Items	0 or 1	0 = no, 1 = yes
Allow Cursor to Enter Window	0 - 2	0 = always 1 = only when active 2 = never
Multiple Column Order	0 or 1	0 = column major, 1 = row major
Maximum Item Label Length	0 - 80	
Item Label Justification	0 - 2	0 = left, 1 = right, 2 = free format
Active Window	0 or 1	0 = no, 1 = yes read only field
Number of Items in Window	0 - N	read only field

Table 3-1 Window-Level Attributes (Continued)

<i>Window Attributes</i>	<i>Value</i>	<i>Description</i>
Active Window Attributes:		
Blinking	0 or 1	0 = no, 1 = yes
Underlining	0 or 1	0 = no, 1 = yes
Reverse Video	0 or 1	0 = no, 1 = yes
Intensity/Color	0 - 7	0 = black (lowest), 1 = blue, 2 = red, 3 = magenta, 4 = green, 5 = cyan, 6 = yellow, 7 = white (highest)
Inactive (Normal) Window Attributes:		
Blinking	0 or 1	0 = no, 1 = yes
Underlining	0 or 1	0 = no, 1 = yes
Reverse Video	0 or 1	0 = no, 1 = yes
Intensity/Color	0 - 7	0 = black (lowest), 7 = white (highest)
Window Label Attributes:		
Invisible Label	0 or 1	0 = no, 1 = yes
Label Position	0 - 3	0 = top, 1 = left, 2 = right, 3 = bottom
Centered Label	0 or 1	0 = no, 1 = yes
Blinking Label	0 or 1	0 = no, 1 = yes
Underlined Label	0 or 1	0 = no, 1 = yes
Reverse Video Label	0 or 1	0 = no, 1 = yes
Label Intensity/Color	0 - 7	0 = black (lowest), 7 = white (highest)
Use Item Intensity	0 or 1	0 = no, 1 = yes
Cursor Attributes:		
Cursor Size	0 - 4	0 = invisible, 1 = single char, 2 = full size, 3 = item and label, 4 = item only
Blinking Cursor	0 or 1	0 = no, 1 = yes
Underlined Cursor	0 or 1	0 = no, 1 = yes
Reverse Video Cursor	0 or 1	0 = no, 1 = yes
Cursor Intensity/Color	0 - 7	0 = black (lowest), 7 = white (highest)
Use Item Intensity	0 or 1	0 = no, 1 = yes
Don't Delete Cursor	0 or 1	0 = no, 1 = yes Can only be set in the application. Screen Builder default is 0.

Table 3-1 Window-Level Attributes (Continued)

<i>Window Attributes</i>	<i>Value</i>	<i>Description</i>
Border Attributes:		
Bordered Window	0 or 1	0 = no, 1 = yes
Border Takes up Space	0 or 1	0 = yes, 1 = no
Display Scroll Markers	0 or 1	0 = no, 1 = yes
Reverse Video Border	0 or 1	0 = no, 1 = yes
Border Intensity/Color	0 - 7	0 = green, 1 = blue, 2 = red, 3 = magenta, 4 = green, 5 = cyan 6 = yellow, 7 = white

Window Position **3.3.1** The window position values specify window column and row coordinates. The NaturalLink Window Manager supports video devices capable of displaying 80 columns and 25 rows of characters. The upper left coordinates of the display are 0 and 0, and the lower right coordinates of the display are 79 and 24. You specify window size and position by entering the upper left and lower right coordinates of the window. If item selection is being done, the window must be long enough to display at least one item. Default coordinates are 0 for the left-most column, 0 for the top row, 79 for the right-most column, and 24 for the bottom row. These coordinates create a window as large as the entire display area.

Window Format **3.3.2** This set of attributes determines how Window Manager will treat the items within a particular window; that is, the attributes determine how the items will be displayed, whether they can be selected, and so forth.

Window Type **3.3.2.1** The values selected in this field designate the type of window. The default value is 0 (list window). Selectable values are as follows:

<i>Value</i>	<i>Type of Window</i>
0	List Window
1	Text Window
2	Edit Window
3	File Window
4	Display Window
10-99	User-Defined Window

The window types are explained in Chapter 2, Window Manager Features; user-defined windows are discussed in Chapter 8, User-Defined Windows.

The file window is used to display the text file specified by the file window pathname. Window Manager reads this file and transforms the text into window items that can be scrolled. Window Manager displays the file window items according to the other window attributes, such as format, intensity, centering, and so on.

Some limitations are put on file window text files:

- The file must fit into memory. If it does not fit, an error is returned to the application program, and if the window is displayed, no items appear.
- Window Manager uses the carriage return character as a delimiter for the items it creates. If there are no carriage return characters in the file, a single item is created. The results of scrolling such a window cannot be guaranteed.
- If a file window pathname is not given for the file, the results are the same as when the file does not fit into memory.
- Items existing for the file window before the file is loaded are deleted when the file is loaded.
- The file text is loaded into memory when an Add or Refresh call is made and unloaded from memory when a Delete call is made. (See Chapter 6, Window Manager Callable Routines, for more information on Window Manager calls.)
- No item manipulation (adding, deleting, setting attributes, and so on) is allowed for file windows.

Number of Columns

3.3.2.2 The maximum number of columns allowed in a window varies with the width of the window. You can have the same number of columns as the window width, excluding the window's vertical borders if the window is bordered (each vertical border takes up one column of space). For example, if a bordered window is 50 columns wide (52 columns counting the borders), you can have up to 50 columns in the window. If the window is unbordered and the border takes up no space, the possible number of columns is 52. The total number of columns specified determines the format of the window. The default value is 1. See paragraph 3.3.7, Border Attributes, for additional information.

A value of 0 (zero) indicates that the items are displayed in free-format style (one right after the other, with one space separating them). If an entire item cannot fit on the same line as the previous item, the entire item is moved down to start the next line. Truncation occurs if a single item is wider than the width of the window.

A value greater than 0 indicates that the items are displayed in columnar format. In single-column windows, items wider than the width of the window become multiple-line items. However, a flag can be set to disable this multiple-line feature. When this flag is set, the item is truncated if it is too wide.

Multiple-column items are displayed in either row-major or column-major order, depending on the value of the Multiple Column Order attribute. (See Appendix B, Helpful Hints, for an explanation of how row-major and column-major can be used in different applications.) Multiple-column items are truncated if they are longer than the width of a column, where column width is equal to the usable window width divided by the number of columns.

Window Priority 3.3.2.3 The values for this attribute range from 0 to 16. This attribute is not used by Window Manager, but can be used by an application program for determining the priority of active windows. The default value is 0.

Multiple-Selection Window 3.3.2.4 A value of 1 for this attribute indicates that this window is a multiple-selection window. A user can select more than one item from a multiple selection window. In a list window with this feature, the **ENTER** key (which is the default key for the select function) acts as a toggle selecting (highlighting) or unselecting (removing the highlighting) an item. When the item is selected or unselected, the Chosen/Enable attribute is set on or off as well. Thus, when the item is selected, it is displayed with the Item-Chosen attributes.

All selections are committed when the user presses the **F10** key. This is the default key for the Window Manager proceed function. The application program must check the Chosen/Enable attribute of the items in the window to determine which ones the user has selected. The application program is also in charge of turning off (setting to 0) the Chosen/Enable attribute.

In a multiple-selection edit window, Window Manager permits you to edit all fields in the window before returning to the calling routine. Pressing the **ENTER** key moves the cursor to the next field. Pressing the **ENTER** key on the last field commits all edits. Pressing the **F10** key commits all edits at any time. Multiple selection has no effect on text, file, display, or user-defined windows. The default value is 0 (no).

Pop-Up Window **3.3.2.5** A value of 1 for this attribute indicates that this window is a pop-up window. There are only two differences between pop-up windows and other windows. One difference is that border characters are not combined with any other border characters; that is, the window is separate from the other windows on the screen. Also, when this window is active and the cursor is in it, the user cannot move the cursor to any other window until processing of the pop-up window is complete. The default value is 0 (no).

Special Repaint on Receive **3.3.2.6** Receive calls must be made from the application program to get user responses to or from a window. All windows marked for repainting are repainted when the Receive call is made. However, repainting all of the flagged windows can be undesirable for certain applications. The Special Repaint on Receive attribute can remedy this. If a window is flagged for repainting and this window is specified in a Receive call, a value of 1 indicates that this will be the only window repainted even if other windows have also been flagged for repainting.

This attribute is not valid for display windows. The default value is 0 (no).

First Item to Be Displayed **3.3.2.7** The first item to be displayed is referred to as the offset value for the list of items. The assigned *offset value* is the first item displayed in the window. Window Manager always starts displaying items with this first item or offset value. This offset value changes as the window is scrolled or if other changes are made to the window that result in a different first item. The default value is 0.

Screen Builder does not permit the offset value to be changed, and the first item in the list (zero offset) becomes the default value for this attribute. However, in your application program you can change this field to have the window painted starting with a different offset value. Setting this attribute value to -1 ensures that the window will be displayed starting with the first visible item.

Show Last Item When Painted **3.3.2.8** A value of 1 for this attribute indicates that when a free-format window is painted, it will be painted so that the last item in the window is showing. This feature is like having an automatic scrolling capability for the window. It ensures that each time the window is painted, the last item will be visible. This feature is available only for free-format windows. The default value is 0 (no).

Center All Items **3.3.2.9** A value of 1 for this attribute indicates that the items in the window are to be centered. Centering is based on the maximum label length plus the length of the item itself. Centering does not apply to free-format windows. The default value is 0 (no).

Don't Redisplay Current Items **3.3.2.10** A value of 1 for this attribute instructs Window Manager to display only new items in the window. In other words, if items are already displayed in the window and the window gets repainted, only the items not already displayed are painted. With this option, Window Manager does not spend a lot of time repainting items whose appearance has not changed. If this attribute is set, then no matter what Window Manager call is made, the only time the window is completely repainted is when the First Item to be Displayed attribute is set to -1. The default value is 0 (no). This attribute is used in conjunction with the Displayed attribute.

Disable Multiple-Line Items **3.3.2.11** Setting the value of this attribute to 1 disables multiple-line items. When this attribute is set, items will be truncated if they are longer than the width of the window, and Window Manager can scroll it much faster than usual. This attribute is used only with single-column windows. The default is 1 (yes).

Allow Cursor to Enter Window **3.3.2.12** This attribute is used for all window types except display windows. It indicates to Window Manager when to allow the cursor to enter the window. Normally the cursor can enter any nondisplay-type window when window movement keys are pressed or when a Receive call is made on the window. There are three options; the default option is 0 (always).

- 0 (Always). There are no limitations as to when the cursor can enter the window. When any window movement key is pressed to move into the window or a Receive call is made on the window, the move will be allowed.
- 1 (Only when active). The cursor will be allowed to enter the window only when the window is active—that is, when the window has been selected.
- 2 (Never). The cursor will never be allowed to enter this window. This results in the equivalent of a display window.

This attribute was designed primarily for use with user-defined windows (UDWs). Since the application controls all cursor movement in a UDW, the application can limit the times this cursor movement can be done. Or the application can use the never option and create a UDW that is like a display window.

Multiple-Column Order **3.3.2.13** This specifies the order in which multiple-column items are displayed and the order in which they are scrolled and selected. Two options are available. The default value is 0 (column-major).

■ Column-major — Top-to-bottom, left-to-right order (value = 0)

■ Row-major — Left-to-right, top-to-bottom order (value = 1).

Maximum Item Label Length **3.3.2.14** The value of this attribute specifies the maximum length of an item label. A value of 0 prevents Window Manager from displaying any item labels, even if text for the item labels has been specified, thus giving you invisible item labels. A value greater than 0 specifies the field width used for item labels in left- and right-label justification. No matter what type of label justification is used, if the item label text is longer than the specified maximum length, the excess label text is truncated. If the maximum item label length is greater than the column width, Window Manager reduces the maximum item label width to equal the column width. The default value is 0 (no item labels).

Item Label Justification **3.3.2.15** Specify 0 for this attribute to left-justify the label within the Maximum Item Label Length field. Specify 1 to right-justify the label within the Maximum Item Label Length field. Specify 2 for free-format item labels. If you specify 2, the item text will immediately follow the item label, and the Maximum Item Label Length value will be ignored if the actual label length is less than this maximum length.

Since the item label is written to the screen first, the item is truncated if the specified column width is not great enough to contain both the label and the item. The default value is 0 (left-justify).

Active Window **3.3.2.16** A value of 1 for this attribute indicates that the window is active. This is a read-only attribute field for the application. This attribute is set to 1 when a Select call is made and to 0 when a Release call is made. (Refer to Chapter 6, Window Manager Callable Routines.) When the value of this attribute is 1, the items in the window are displayed using the active attributes. When the value is 0, the items are displayed using the inactive attributes. The default value is 0 (no).

Number of Items in Window **3.3.2.17** This is a read-only attribute field for the application program. This attribute indicates the current number of items in the window. This number includes every item regardless of whether the item is invisible, unselectable, or has no text. The default value is 0 (no items).

**Active
Window
Attributes**

3.3.3 The following attributes are in effect when the window is active.

- **Blinking** — A value of 1 causes every character to blink. The default value is 0 (no).
 - **Underlined** — A value of 1 underlines every character. The default value is 0 (no).
 - **Reverse video** — A value of 1 displays every character in reverse video. The default value is 0 (no).
 - **Intensity/color** — Values from 0 to 7 set eight levels of monochrome intensity or color. The default value is 7 (white).
-

**Inactive
Window
Attributes**

3.3.4 The following attributes are in effect when the window is inactive.

- **Blinking** — A value of 1 causes every character to blink. The default value is 0 (no).
 - **Underlined** — A value of 1 underlines every character. The default value is 0 (no).
 - **Reverse video** — A value of 1 displays every character in reverse video. The default value is 0 (no).
 - **Intensity/color** — Values of 0 to 7 set eight levels of monochrome intensity or color. The default value is 4 (green).
-

**Window
Label Attributes**

3.3.5 Window label attributes are used to control the way the window label is displayed.

Invisible Label

3.3.5.1 A value of 1 for this attribute indicates that the window label is not visible (not displayed in the window). This attribute is useful when you want the window label to identify the window in the screen file list but you do not want it to show when the window is displayed. The default value is 0 (no).

Label Position 3.3.5.2 This attribute specifies the position of the window label within the window. The default value is 0 (top). Selectable values are as follows:

<i>Value</i>	<i>Label Position</i>
0	Top of Window
1	Left Side of Window
2	Right Side of Window
3	Bottom of Window

If Top of Window or Bottom of Window is chosen, the label occupies either the top line(s) or bottom line(s) of the window, respectively; no item can share the same line(s) with the label. If Left Side of Window or Right Side of Window is chosen, the label occupies the corresponding side of the first line(s) of the window; in this case, all items are shifted left or right (flush left or flush right), and the items can share the same line(s) as the label. If the window label occupies multiple lines, each line will be placed as far to the left as possible for a left label position and as far to the right as possible for a right label position. For uncentered top and bottom labels, all label lines are left-justified.

Centered Label 3.3.5.3 A value of 1 for this attribute indicates that the label is to be centered. This attribute is used only if the label is on the top or bottom of the window. The default value is 0 (no). If the label has multiple lines, each line is centered independently of one another.

Blinking Label 3.3.5.4 A value of 1 for this attribute causes the window label to blink. The default value is 0 (no).

Underlined Label 3.3.5.5 A value of 1 for this attribute underlines the window label. The default value is 0 (no).

Reverse Video Label 3.3.5.6 A value of 1 for this attribute displays the label in reverse video. The default value is 0 (no).

Label Intensity/Color 3.3.5.7 One of eight levels of monochrome intensity or color can be set for this attribute. The default value is 7 (white).

Use Item Intensity 3.3.5.8 A value of 1 for this attribute indicates that the window label is to be displayed with the same intensity as the items in the window. This would be an active or inactive intensity, depending on whether the window is active or not. The default value is 0 (no).

Cursor Attributes **3.3.6** You can use cursor attributes to specify different types and sizes of cursors. The combination of these attributes determines the overall appearance of the cursor. For example, if you specify the attributes for a full-size reverse video cursor, a reverse video cursor which spans the entire width of a column is displayed.

Size and Type **3.3.6.1** The value of this attribute field specifies the size and type of a cursor. Default values for size/type cursor attributes force the cursor to be invisible in text and file windows, and single-character size in edit windows. The default value for list windows is 2 (full size). Selectable values are as follows:

<i>Value</i>	<i>Size/Type</i>
0	Invisible
1	Single character size
2	Full size (spans entire column width)
3	Label and item size (spans only item label and item)
4	Item size (spans item only)

Blinking Cursor **3.3.6.2** A value of 1 for this attribute causes the cursor to blink. The default value is 0 (no).

Underlined Cursor **3.3.6.3** A value of 1 for this attribute results in an underlined cursor. The default value is 0 (no).

Reverse Video Cursor **3.3.6.4** A value of 1 for this attribute displays the cursor in reverse video. The default value is 1 (yes).

Cursor Intensity/Color **3.3.6.5** Values of 0 to 7 for this attribute set the levels of monochrome cursor intensity or color. A single-character cursor can assume only the intensity of the character over which it is positioned. The default value is 5 (cyan).

Use Item Intensity **3.3.6.6** This attribute sets an alternate intensity for the cursor. The cursor, other than the single-character cursor, can have either its own unique intensity or the same intensity as the item over which it is positioned. A value of 1 indicates that it will share the same intensity as that of the item the cursor is positioned over. The default value is 0 (no).

Don't Delete Cursor **3.3.6.7** An application receives a user's response to items in a window by making a Receive call. Window Manager returns to the application program when **ENTER**, **F10**, or any undefined key is pressed. When a Receive call is made, Window Manager puts the cursor in the window and, upon returning to the application, deletes the cursor. Displaying and deleting the cursor causes the cursor to flash if the application is looping until **ENTER** or **F10** is pressed. To stop this flashing, you can set the Don't Delete Cursor attribute to 1. This indicates that Window Manager should not delete the cursor when returning from a Receive call unless **ENTER** or **F10** has been pressed. (This is only true when looping on a Receive call. If the window is redisplayed in some manner between receive calls, the action of the cursor cannot be guaranteed.) The default value is 0 (no).

Border Attributes **3.3.7** Border attributes can be set for each window. Care should be taken when choosing border attributes, as the same border attributes will be used whether the window is active or inactive. Also, the use of different border attributes for windows which share borders may not have the desired outcome. Even though the borders themselves are combined, their attributes are not. The common border will always retain the display attributes of the last window painted.

Bordered Window **3.3.7.1** A value of 1 for this attribute indicates that this window has a border. Character graphics are used to draw the border. The default value is 1 (yes). A space is reserved for the border whether or not the border is displayed. Thus, if the window coordinates are 0,0,79,24, the maximum length of a displayed character string on one window line is 78 characters.

Border Takes Up Space **3.3.7.2** When a window is unbordered, Window Manager will still reserve space on the display for the border. If you would like to use this space, set the Border Takes Up Space attribute to 1; this causes all text in the window to start in the space normally reserved for the border. Thus, lines of text can be up to 80 characters in length. Scroll markers cannot be used with this window. If the attribute is set to 1 when the window is bordered, the border will be cleared when text is put into the window. The default is 0 (Border Takes Up Space). Note that the values 0 and 1 are used in the opposite manner of the other calls.

Display Scroll Markers **3.3.7.3** A value of 1 for this attribute indicates that scroll markers should be placed in the middle of the bottom border. The markers, two small arrows, are used to indicate that there are more items to view: an up arrow indicates that there are more items at the top of the window; a down arrow indicates that there are more items at the bottom. The default value is 0 (no).

Reverse Video Border **3.3.7.4** A value of 1 for this attribute causes the border to be displayed in reverse video.

Border Intensity/Color

3.3.7.5 One of seven different levels of monochrome intensity or color can be set for this attribute. Note that a green (intensity level 4) border will be displayed when this attribute is set to either 0 or 4. The default value is 4 (green).

Item-Level Attributes

3.4 Each item has many attributes that describe it. The attributes determine how Window Manager will treat the items. As with window-level attributes, item-level attributes can have character string values as well as numeric values.

Item-Level Attributes with String Values

3.5 String value attributes at the item level can be used to label an item, request the user to enter data, display textual material, and so forth.

Item Label

3.5.1 Every item in the window can have a label. Item labels are especially useful for prompts in edit fields. They can be up to 80 characters in length (if the window border takes up no space) and are truncated if they are longer than the value specified for the Maximum Item Label Length. If the Maximum Item Label Length is 0, the item label will not appear in the window, even if item label text was specified.

Item Text

3.5.2 The text of a single-line item can be up to 80 characters in length. Truncation occurs if text is too wide to fit within the window boundaries. If truncation occurs, the item text is truncated first and the item labels are truncated second. For multiple-line items, word wrapping occurs if the item text does not fit on a single line within the window boundaries.

In a list, text, or display window with columnar format, an item that has no text shows up as a blank line in the window. To get the same results in an edit window with columnar format, there must be no text specified and the Maximum Edit Field Length attribute for the item must be set to 0. If the item should not appear at all, the Visible Item attribute (refer to Table 3-2) must be set to 0. In a free-format window, items without text are not visible and occupy no space in the window.

Item-Level Attributes with Numeric Values

3.6 Most item-level attributes can be assigned numeric values. There are two groups of attributes:

- Item format attributes, which are in effect all the time
- Item-chosen attributes, which are effective only when the Chosen/Enable attribute is set.

Table 3-2 presents a list of these attributes. The paragraphs following the table contain a description of each.

Table 3-2**Item-Level Attributes**

<i>Field Description</i>	<i>Value</i>	<i>Description</i>
Item Format Attributes:		
Visible Item	0 or 1	0 = no, 1 = yes
Unselectable Item	0 or 1	0 = no, 1 = yes
Displayed	0 or 1	0 = no, 1 = yes Can be set only within the application. Screen Builder defaults to 0
Maximum Edit Field Length	0 - 80	
Edit Field Datatype	0 - 200	0 = no validation done 1-200 = application-defined
Required Edit Field	0 or 1	0 = no, 1 = yes
Echo Edit Field Input	0 or 1	0 = no, 1 = yes
Chosen/Enable Attributes	0 or 1	0 = no, 1 = yes
Item-Chosen Attributes:		
Blinking Chosen Item	0 or 1	0 = no, 1 = yes
Underlined Chosen Item	0 or 1	0 = no, 1 = yes
Reverse Video Chosen Item	0 or 1	0 = no, 1 = yes
Chosen Intensity/Color	0 - 7	0 = black (lowest), 7 = white (highest)
Item Label Attributes:		
Blinking Label	0 or 1	0 = no, 1 = yes
Underlined Label	0 or 1	0 = no, 1 = yes
Reverse Video Label	0 or 1	0 = no, 1 = yes
Label Intensity/Color	0 - 7	0 = black (lowest), 7 = white (highest)
Use Item Intensity	0 or 1	0 = no, 1 = yes

Item-Format Attributes **3.6.1** Item-format attributes are permanent item attributes. These attributes are in effect at all times.

Visible **3.6.1.1** A value of 1 for this attribute makes the item visible. An item may be in the item list but will not appear on the display unless this attribute value is set to 1. The default value is 1 (yes).

Unselectable **3.6.1.2** Setting a value of 1 for this attribute makes the item unselectable. An unselectable item appears on the display, but the cursor cannot be positioned over the item. Extensive use of this field can produce some cursor placement problems; difficulties can occur if there is not at least one selectable item in the window at all times. If an entire window of unselectable items is desired, the display window type should be used, or the Allow Cursor to Enter Window attribute should be set to Never (0). The default value is 0 (no).

Displayed **3.6.1.3** This attribute is used only when the value of the Don't Redisplay Current Items attribute is set to 1. Window Manager automatically sets the value of the Displayed attribute to 1 after it displays the item. Thus, if the application has set the Don't Redisplay Current Items attribute but requires a particular item to be redisplayed, the value of the Displayed attribute must be set to 0 for that particular item. This is useful when you want to redisplay a highlighted item but you want to prevent the momentary blink or screen flash that results from redisplaying all of the items. The default value is 0 (no).

Maximum Edit Field Length **3.6.1.4** This attribute only pertains to edit windows. The value assigned to this attribute determines the maximum number of characters that can be entered for the item. If this value is 0 but the item is still selectable, Window Manager does not allow any characters to be entered for the item. The default value is 0.

Edit Field Datatype **3.6.1.5** An application can perform datatype checking on any item in an edit window. When the cursor is moved off an edit field by pressing the **ENTER**, **F10**, cursor movement, or any unknown keys, Window Manager calls the application's validation routine. The application uses the datatype field to identify the type of validation to be done on the field. The application designer defines all values. The assignable datatypes are 0 through 200. A value of 0 means Window Manager will *not* call the application's validation routine. The default value is 0 (no validation done). See Chapter 7, Application Validation Routine, for a full discussion of the edit field validation process.

Required Edit Field **3.6.1.6** A value of 1 for this attribute indicates that a user must enter a value in a window. This is only used for edit windows. Window Manager checks this attribute only when the **ENTER** key (to select item) is pressed or when the **F10** key (to move out of a multiple-selection window) is pressed. The default value is 1 (yes).

Echo Edit Field Input **3.6.1.7** A value of 1 in this field causes input into an edit field to be echo-printed to the screen. This attribute pertains only to edit windows. It can be used to prevent the display of sensitive information, such as passwords. If the attribute has a value of 0 when the user enters data, the cursor will move through the edit field, but no characters will be echo-printed to the screen. The default value is 1 (yes).

Chosen/Enable Attribute **3.6.1.8** This attribute is used primarily with multiple-selection list windows. It is set to 1 when the item has been selected and must be checked by the application program to determine the items chosen from this window. When this attribute is set to 1, the item appears using the item-chosen attributes. If not set (value = 0), the way the item appears depends on whether the window is active or inactive.

The Chosen/Enable attribute can also be used to achieve different effects in the window; that is, the application program can set it whenever an item should appear with different attributes than those in the rest of the window. The default value is 0 (no).

Item-Chosen Attributes **3.6.2** The item-chosen attributes are used only when the Chosen/Enable attribute is set.

Blinking Chosen Item **3.6.2.1** A value of 1 for this attribute causes the chosen item to blink. The default value is 0 (no).

Underlined Chosen Item **3.6.2.2** A value of 1 for this attribute underlines a chosen item. The default value is 0 (no).

Reverse Video Chosen Item **3.6.2.3** A value of 1 for this attribute displays a chosen item in reverse video. The default value is 1 (yes).

Chosen Item Intensity/Color **3.6.2.4** A value of 0 to 7 sets eight levels of monochrome intensity or color for the chosen item. The default value is 7 (white).

Item Label Attributes **3.6.3** These attributes determine the display characteristics of the item label.

Blinking Label **3.6.3.1** A value of 1 for this attribute causes an item label to blink. The default value is 0 (no).

Underlined Label **3.6.3.2** A value of 1 for this attribute underlines an item label. The default value is 0 (no).

Reverse Video Label **3.6.3.3** A value of 1 for this attribute displays the item label in reverse video. The default value is 0 (no).

Label Intensity/Color **3.6.3.4** One of eight levels of item-label intensity or color can be set for this attribute. The default value is 7 (white).

Use Item Intensity **3.6.3.5** A value of 1 for this attribute indicates that the item label should be displayed using the same intensity as the item itself. The default value is 0 (no).

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
4.1	Using Screen Builder	4-3
4.2	Screen-Level Commands	4-4
4.2.1	Edit Window	4-4
4.2.2	Add Window.....	4-5
4.2.3	Copy Window.....	4-5
4.2.4	Insert Window.....	4-5
4.2.5	Delete Window.....	4-5
4.2.6	Change Help File.....	4-5
4.2.7	Draw Screen.....	4-6
4.2.8	Test Screen.....	4-7
4.2.9	Load Screen.....	4-7
4.2.10	Save Screen.....	4-8
4.2.11	List Screen Attributes.....	4-8
4.2.12	Exit	4-8
4.3	Window-Level Commands.....	4-9
4.3.1	Set Window Position	4-10
4.3.2	Set Window Format	4-10
4.3.3	Set Inactive Window Attributes.....	4-10
4.3.4	Set Active Window Attributes	4-10
4.3.5	Edit Window Label.....	4-11
4.3.6	Set Window Label Attributes	4-12
4.3.7	Set Cursor Attributes.....	4-12
4.3.8	Set Border Attributes	4-12
4.3.9	Attach Help to Window.....	4-12
4.3.10	Edit Item Table	4-13
4.3.11	Draw Window	4-13
4.3.12	Test Window	4-13
4.4	Item-Level Commands	4-13
4.4.1	Edit Item Text.....	4-14
4.4.2	Set Item Attributes	4-15
4.4.3	Edit Item Label	4-15
4.4.4	Set Item Label Attributes	4-16
4.4.5	Add, Copy, Insert, and Delete Item	4-16

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
4.4.6	Attach Help to Item.....	4-16
4.5	The Help File	4-16
4.5.1	Changing the Help File	4-16
4.5.2	Setting a Path for the Help File	4-17
4.6	Attaching Help to Windows and Items	4-18
4.6.1	Add Help Message to List.....	4-19
4.6.2	Insert Help Message Into List	4-20
4.6.3	Delete Help Message From List	4-20
4.6.4	View Help Message	4-20
4.6.5	Specify Help Message.....	4-20

Using Screen Builder

4.1 Screen Builder aids in constructing screens for use with Window Manager routines. The interactive menus in the Screen Builder utility are used to set attributes described in Chapter 3 (Window Attributes). Refer to that chapter for specific information about attributes. If you need help while using Screen Builder, press the F7 key to display online Help messages.

The SBUILD command invokes the Screen Builder utility. Before entering the utility, you should first use the MS-DOS SET command to set up the environment variable NLXTOOLS. NLXTOOLS must be set to the directory pathname in which the screen files and message files for SBUILD exist (the .PIC, .NS\$, and .NM\$ files). If NLXTOOLS is not set, SBUILD will expect these support files to be on the default drive and directory. If you are using a pre-2.0 version of the operating system, ignore NLXTOOLS and make sure Screen Builder support files are on the default drive.

NOTE: All Window Manager utilities use the environment variable NLXTOOLS to indicate the location of their support files. Therefore, it would be best to place support files for all of the utilities on the same directory.

If the file pathname of a screen to be loaded is included on the command line following the SBUILD command, the screen file will be loaded at the start of the Screen Builder utility. If this screen file does not exist or SBUILD has problems loading the screen, a message will be displayed indicating the problem before Screen Builder continues.

Screen-Level Commands

4.2 When you enter SBUILD, the main menu for the Screen Builder utility (Figure 4-1) is displayed. If no screen has been loaded and/or no windows are added, only the Add Window, Load Screen, and Exit commands are available. Once a window has been added or a screen loaded, the rest of the commands appear. These commands are explained in the following paragraphs.

Figure 4-1

Screen Builder Main Menu — Screen Commands

NATURALLINK SCREEN BUILDER UTILITY	
COMMANDS	
Edit Window	Draw Screen
Add Window	Test Screen
Copy Window	Load Screen
Insert Window	Save Screen
Delete Window	List Screen Attributes
Change Help File	Exit
WINDOWS	
0	NaturalLink Access to ...
1	< no label specified >
2	Main Menu:
3	Press:
Select a command	

Edit Window

4.2.1 To set or make any modifications to the window or item attributes, the Edit Window command must be used. When the Edit Window command is chosen, the cursor is placed in the `WINDOWS` window, which lists all the windows associated with the current screen. When one of these windows is selected, the Window-Level Menu (see Figure 4-2 later in this chapter) is displayed to allow you to set window-level attributes. The edit choice can be aborted by pressing the `ESC` key before a window is selected. The cursor then returns to the `COMMANDS` window.

Add Window 4.2.2 When creating a new screen file or adding windows to an existing file, the Add Window command must be used. Add Window creates a new window with all of the window attributes set to their default values. If you are adding windows to a previously created screen, that screen must first be loaded into SBUILD.

For each window added, a new window entry will be created at the end of the list of windows displayed in the lower portion of the Screen Builder Main Menu. Each new window is shown initially as a number and window label. Windows that do not have window labels are listed as <no label specified>. The accompanying number indicates the window index in the screen structure. This window number will be used as input to the majority of the Window Manager calls.

Copy Window 4.2.3 This command enables you to copy a previously specified window. After Copy Window is selected, the cursor will be positioned on the list of window names. The name of the master window should be selected first, then the cursor should be moved to the name of the destination window. When the destination window is selected, it is assigned the items, attributes, and help numbers of the master window. If attributes have already been specified for the destination window, they are replaced with the new attributes. Pressing the **ESC** key before selecting the destination window aborts the reassignment procedure.

Insert Window 4.2.4 This command allows a new window to be inserted into the list of windows. The attributes of the newly created window will all be set to their default values. When you select Insert Window, the cursor is placed at the list of windows. A new window will be inserted above the selected item. If you want to abort the procedure, press the **ESC** key before selecting the window position.

Delete Window 4.2.5 This command deletes a window from the window list. When a window is selected, it is deleted, and the other windows in the list are renumbered. Pressing the **ESC** key before the window is selected aborts the operation.

Change Help File 4.2.6 This command is used to change the help file pathname associated with a screen. Pressing the **ESC** key aborts this operation. More information about the help file is presented later in this chapter.

Draw Screen 4.2.7 This command allows you to see all or a subset of the windows you have created without having to write any application code. When this command is selected, a pop-up window is presented that asks you to choose which windows in the current screen file you wish to draw.

Any one of the following options can be selected.

- **ALL OF THEM** — With this choice, all of the windows listed will be drawn.
- **NON-POPUPS ONLY** — This choice will only draw windows which have the pop-up attribute set to 0 (no).
- **A SELECTED SUBSET** — This allows you to select a subset of the windows listed to be drawn. Use the **ENTER** key to select the windows you wish drawn. Press the **F10** key, and the selected windows will be drawn.

When the windows are drawn, they are shown in their inactive state; that is, the inactive attributes are used. The Draw Screen command is actually performing an Add call followed by a Display call on all the desired windows. (Refer to Chapter 6, Window Manager Callable Routines.) Therefore, they will be drawn in order of their index in the screen file. If you want to obtain a printout of the drawn windows, press the **CTRL-PrtSc** keys. If you do not want a printout, press any key except **CTRL-PrtSc** to return to the main menu.

If the **CTRL-PrtSc** keys are pressed, a prompt requests the location to which you wish a copy of the screen to be sent—either to the printer or to a file. If you select a file, you must enter a valid operating system path-name. A text file with the screen picture in it is then created. This print capability applies only to borders and text. Attributes such as intensity or reverse video do not show up in the printout. If you want a list of all the text and attributes in the screen, use the List Screen Attributes option. (See paragraph 4.2.11.)

Test Screen 4.2.8 The Test Screen command permits the testing of actual cursor movement and display of help information for all windows being tested. As with the Draw Screen command, you have the same choices as to which windows listed will be tested. Once the windows have been displayed, the cursor can be moved around this screen to test the features of all the windows. You can test the Help function for any window as well. You can also get a printout of the screen in the same manner as in the Draw Screen command. After making a selection from any of the windows, the Screen Builder main menu reappears.

When the screen is tested, all of the windows shown are in their active state; that is, the active attributes are used. Screen Builder implements this feature by making Add and Select calls on all of the specified windows. The cursor is positioned on the first window in the list by means of a Receive call on that window. Since all windows are active, an item can be selected from any valid window. (Refer to Chapter 6, Window Manager Callable Routines.) You can test other windows in the screen by pressing the **CTRL-HOME** keys, the **CTRL-Pg Up/Dn** keys, or the **CTRL-left/right arrow** keys to move the cursor into other windows.

NOTE: If only one window of several selected is displayed on a Test Screen operation, check the value of the Special Repaint on Receive attribute for the displayed window. When the value of this attribute is 1 (yes), only this single window will be displayed.

Load Screen 4.2.9 This command loads a screen that has been previously saved to the disk. It presents a pop-up window in which you enter the pathname of the screen file you want to load into Screen Builder. If there are any windows currently being shown in the **WINDOWS** window, they are destroyed when the screen is loaded. The Load Screen command should be used when you want to modify or add to an existing screen file. The loading operation can be aborted by pressing the **ESC** key.

Save Screen 4.2.10 This command saves all the listed windows as one screen file on a disk. They are saved in a screen structure in the order in which they are listed. A pop-up window is presented in which you enter the pathname of the file in which you want the current screen saved. The screen is saved in the pathname specified. If the specified file already exists, it will be overwritten by the new file. Once the screen has been saved, only the Add, Load, and Exit commands will be available again, since saving the screen clears out all windows listed. Pressing the **ESC** key aborts the operation.

List Screen Attributes 4.2.11 This option will produce a text file listing of the attributes for all the windows in the screen you are currently working on. This file lists the name of the help file for the screen, the window label, and all the attributes currently set for each window. For each item in a window, the item label, item text, and all the attributes currently set for each item are listed.

After List Screen Attributes is selected, a pop-up window prompts for the pathname of the file where the listing should be stored. Once the pathname has been specified, you are asked whether you want all attributes listed or just the text. Selecting all attributes will cause both the text and the currently set attributes to be listed to the file. Specifying just the text will cause only the help pathname, window labels, item labels, and item text to be listed to the file.

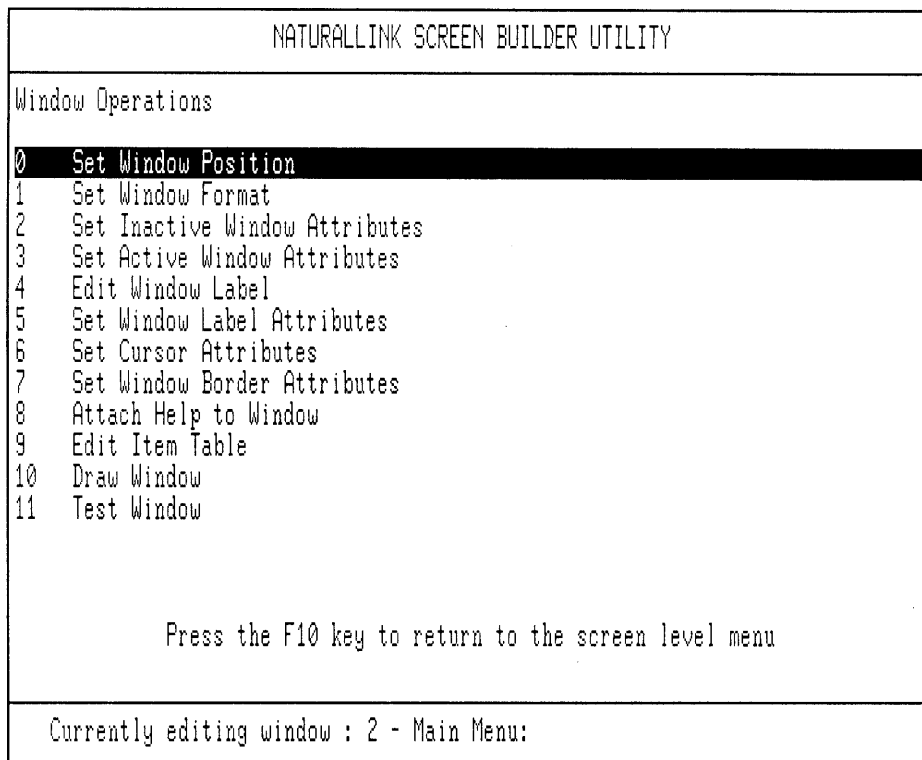
Exit 4.2.12 Selecting this option terminates Screen Builder. If any windows are still specified at this time, a prompt asks if they should be saved before the program terminates. You can abort the operation by pressing the **ESC** key when the pop-up window is displayed.

Window-Level Commands

4.3 Choices from the Window Level Menu (Figure 4-2) trigger a series of pop-up windows that request such information as whether items should be centered, in reverse video, in high intensity, with or without labels, and so on. Also at this level, as well as at the Screen-Command level, you have draw and test options that enable you to see how each window looks and how it behaves.

Figure 4-2

Window-Level Menu



The screen shown in Figure 4-2 contains 12 options (operations) for specifying the attributes of the window. The window identified at the bottom of the screen indicates which window is being edited. Windows that do not have window labels will be referenced by the window index number. Otherwise, the windows will be referenced by the window index number and the window label.

With the exception of Edit Window Label and Edit Item Table, all of the options have default values listed in the windows that appear after you select a given option. (These are explained with each option.)

All but the last four options in Figure 4-2 use multiple-selection pop-up windows in which you enter the information for that option. None of the fields in these pop-ups is required, so the **F10** key can be pressed at any time to exit. To return to the Screen Builder main menu, press the **F10** key.

Set Window Position 4.3.1 This option enables you to specify the upper left and lower right coordinates of a window. Default values produce a window the size of the entire video screen.

Set Window Format 4.3.2 Most of the window attributes are set using this format option. Default values in the Set Window Format menu produce a window with the following characteristics:

- Window Type — List window
 - Column Format — Single column
 - Disable Multiple Line Items — Yes
 - Multiple Selection Window — No
 - Allow Cursor to Enter Window — Always
 - Pop-up Window — No
 - Special Repaint on Receive — No
 - Don't Redisplay Current Items — No
 - Center All Items — No
 - Visible Item Labels — No
 - Set Window Priority — Currently 0
-

Set Inactive Window Attributes 4.3.3 Selecting this option enables you to set the attributes of the items displayed when the window is inactive. The default value displays items in intensity-level 4 (green).

Set Active Window Attributes 4.3.4 Select this option when you wish to set the attributes of the items displayed when the window is active. The default value displays items in intensity-level 7 (white).

Edit Window Label

4.3.5 When this option is selected, a pop-up window is presented in which you can enter up to 25 lines, each 78 characters in length, of window label text. Each line entered in the pop-up window will become a separate line of the window label. Special editing features have been added to this window and are documented with Help messages attached to the window. The editing keys include the following:

- **INS** — Enables you to insert a character or a string of characters.
- **DEL** — Deletes a single character.
- **CTRL-F1** — Moves the cursor to the beginning of a line of text.
- **CTRL-F2** — Moves the cursor to the end of a line of text.
- **CTRL-F3** — Moves the cursor left one word at a time.
- **CTRL-F4** — Moves the cursor right one word at a time.
- **CTRL-F5** — Deletes text from the cursor position to the end of the line. If the cursor is at the first character position, **CTRL-F5** removes the entire line from the window.
- **CTRL-F6** — Inserts a new line into the label text. A new line cannot be inserted if the last line in the window contains text. There is a limit of 25 lines per window label.
- **F5** — Joins the line below the cursor with the line containing the cursor.
- **F6** — Breaks the line at the cursor and places the remainder onto a new line below the one containing the cursor. Since there is a limit of 25 text lines for a window label, the break operation cannot be performed if line 25 has text in it.

Blank lines can be part of the window label. To leave a blank line within the label text, leave the edit line blank. However, if you want to leave one or more blank lines at the end of the window label text, ensure that the last line to be blank has a single space entered as the first and only character of that edit line.

When a window label with blank lines inserted is edited again, all blank lines will show up as a line with a single space as the first character. This is normal and is just another way to reserve a blank line. Any blank line in the window label will not take on any of the window label attributes (that is, it will never be underlined or shown in reverse video). If you want the blank lines to take on attributes, you will have to pad the line with enough spaces to get the desired effect.

When the window label is used as an item in Screen Builder (such as the `WINDOWS` window in the main menu or the bottom window in the window-level menu), a multiple-line window label will be shown as one line with a vertical bar (|) indicating the end of each line of the label.

**Set Window
Label Attributes**

4.3.6 This option is selected when you wish to set the attributes of the window label. The default value causes the label to be displayed left-justified in intensity-level 7 (white) at the top of the window.

**Set Cursor
Attributes**

4.3.7 The cursor can be specified by selecting this item. The default value specifies a full size, reverse video cursor, displayed in intensity-level 5 (cyan).

**Set Border
Attributes**

4.3.8 The Set Border Attributes option allows you to change the appearance of the window border. The default border attributes result in a border which will be drawn in intensity-level 4 (green) and have no scroll markers associated with it.

If you decide to use different border intensities or a reverse video border for adjoining windows in the same display, keep in mind that these two display attributes will not be combined with the border attributes of surrounding windows. If two windows share a common border, the intensity and reverse video border attributes of the window which was drawn last are used for the shared border.

**Attach
Help to Window**

4.3.9 Selecting this item causes the Attach Help Messages screen to appear. This screen enables you to attach Help messages to the window. This option is not available for display windows since the cursor can never enter a display window. (See paragraph 4.6, Attaching Help to Windows and Items, for more information.)

Edit Item Table 4.3.10 Selecting this item causes the Item-Level Menu (Figure 4-3) to appear. On this screen you can specify item-level information such as item text and item attributes.

Draw Window 4.3.11 This option is just like the Draw Screen command on the screen-level menu, except that it only draws the window currently being edited. A printout is not provided at this level.

Test Window 4.3.12 This command is identical to the Test Screen command on the screen-level menu, except that this option tests only the window currently being edited. A printout is not provided at this level. Windows that cannot be tested include the following:

- Display windows
 - Windows with no items
 - Windows with all unselectable items
 - Windows with the Allow Cursor to Enter Window attribute set to Never
 - User-defined windows
-

Item-Level Commands

4.4 The Item-Level Menu (Figure 4-3) is similar to the screen-level menu. The `COMMANDS` window contains a list of the possible operations (commands) that can be performed on the items in the window. The process of selecting commands is similar to the screen level, except that these operations act on the item level rather than the screen level. In the `ITEMS FOR WINDOW` window, if the item has no text, `<no text specified>` is listed for the item. The numbers before the items indicate the item index in the window structure. This item number will be used as input to several Window Manager calls.

If no items have yet been specified, Add Item is the only legal choice. Once you have added an item, the other options appear. The commands are explained in the paragraphs following Figure 4-3. To return to the window-level menu, press the **F10** key.

Figure 4-3

Item-Level Menu

NATURALLINK SCREEN BUILDER UTILITY	
COMMANDS	
Edit Item Text	Add Item
Set Item Attributes	Copy Item
Edit Item Label	Insert Item
Set Label Attributes	Delete Item
Attach Help to Item	
ITEMS FOR WINDOW : 2 - Main Menu:	
0	< no text specified >
1	Build Questions
2	Get Saved Questions
3	Use Terminal Mode
4	Select or Change Data Storage Options
5	< no text specified >
6	Create or Change User Profile
7	Create or Change Communications Profile
8	Take the Tutorial
9	< no text specified >
10	Quit
Select a Command. Press the F10 key to return to the window level menu.	

Edit Item Text

4.4.1 If you select this option, the cursor is positioned at the items portion of the menu. Select the item whose text you wish to edit. When the pop-up window appears, you can enter up to 78 characters of item text. If item text already exists for the item, the existing text is the default text and appears in the pop-up window. If you edit the existing text but decide not to use the new text, press the **ESC** key. This aborts the operation.

Set Item Attributes

4.4.2 After making this selection, the cursor is positioned at the items portion of the menu. Select the item whose attributes you wish to set. The cursor is moved to a menu in which attributes are set or selected for the items. Default values in the item-level menu produce items with the following attributes:

- Visible item — Yes
- Unselectable item — No
- Maximum edit field length — 0
- Edit field datatype — 0
- Required edit field — Yes
- Echo edit field input — Yes
- Chosen/Enable attributes — No

If the Chosen/Enable attribute value is changed to Yes, the following additional default attributes are in effect:

- Reverse video chosen item — Yes
- Chosen intensity/color — 7 (white)

You can press the **F10** key at any time to return to the **COMMANDS** window. Pressing the **ESC** key before selecting an item aborts the Set Item Attributes operation.

Edit Item Label

4.4.3 This option enables you to enter item label text or to edit existing item label text. When this choice is selected, the cursor is positioned at the items portion of the menu. Select the item whose label you wish to edit (or press the **ESC** key to abort). A pop-up window appears in which you can enter up to 78 characters of item label text. If an item label already exists for the item selected, its text is the default text shown in the pop-up window.

**Set Item
Label Attributes**

4.4.4 This option permits you to set the attributes of the item label. When this choice is made, the cursor is positioned at the items portion of the menu. Select the item whose label attributes you wish to modify. Pressing the **ESC** key before selecting the item aborts the operation. The item label for an item is displayed in intensity level 7 (white) by default.

**Add, Copy,
Insert, and
Delete Item**

4.4.5 These commands work in exactly the same manner as those at the screen level, except that they work on items instead of windows.

Attach Help to Item

4.4.6 Selecting this option causes the Attach Help Messages screen to appear. This screen allows you to attach Help messages to the item. The option is not available for display windows. See paragraph 4.6, Attaching Help to Windows and Items, for more information about attaching Help to windows and items.

The Help File

4.5 For every screen created by the Screen Builder utility, there may be a file of Help messages associated with the screen. The first time you wish to attach Help to a window or item, a prompt appears requesting you to enter the pathname of the file which will contain the Help messages associated with this screen. This pathname is then stored in the structures referenced by Window Manager. If at some point you find it desirable to change the name of the file associated with this screen, you can select the Change Help File option from the Screen Builder main menu. The same Help message file can be associated with more than one screen.

**Changing
the Help File**

4.5.1 You can change the Help file associated with a screen by selecting the Change Help File option from the Screen Builder main menu. Normally, you should change the Help filename if you wish to rename the file or if you want to select a Help file from a different drive than the one previously specified. For example, if you specify A:HELP.TXT for the Help file associated with the screen and then choose to move the Help file to the default drive, you must select the Change Help File option and respecify the Help filename HELP.TXT for the default drive.

Changing the Help file does not affect the help numbers previously attached to the windows and items of the screen unless the Help filename is deleted. If the Help filename is deleted in the Change Help File option, a pop-up window prompts that all attached Help message numbers will be removed. The pop-up window asks if all the Help message numbers should be deleted. If Yes is selected, all Help numbers are removed. If No is selected, another prompt requests a new help filename. If a different file is entered as the new Help file, the Help numbers previously attached to the windows and items will now correspond to the new file.

Setting a Path for the Help File

4.5.2 When the Help key is pressed (default: **F7** key), Window Manager references the Help file by using the name of the Help file stored with the screen structure. If you know where the Help file will ultimately reside on the disk, the pathname of the Help file can easily be specified in Screen Builder. However, when the screen is designed, the location of the Help file on the disk may not be known. To solve the problems of this latter example, you can take advantage of the Window Manager path variable, `Llwmpath` (`wmpath` for FORTRAN users).

`Llwmpath` is a string variable which may be set using a Set String Value call. (See Chapter 6, Window Manager Callable Routines, for a full discussion of all Window Manager calls.) If only the help filename is specified in Screen Builder (no drive or directory pathname), the value of the `Llwmpath` variable will be appended to the help filename to get the total help pathname. This appended pathname indicates the file from which Window Manager will get the Help message.

Thus the application could prompt the user for the path where the help file was put on disk and then set `Llwmpath` to this value. Another method can be for the application to make use of an environment variable and then set `Llwmpath` equal to the value of the environment variable.

The `Llwmpath` variable can be set any time. `Llwmpath` will be appended to all of the help filenames in the same application as long as the help filename stored with the screen does not include a drive or directory pathname. `Llwmpath` should contain only the value of a directory pathname; a filename should not be included.

Attaching Help to Windows and Items

4.6 Any number of Help messages can be attached to the items or windows in a screen. When you are interacting with a screen during an application program, you can move the cursor to a particular item and press the **F7** key to display the first Help message attached to that item. All other Help messages for that item can be viewed by pressing the **F7** key to view the next message in the sequence, or by pressing the **F8** key to view the previous Help message. Pressing the **ENTER** key at any time will return you to the screen. If an item has no attached Help message, the Help messages attached to the window are displayed in sequence. If no Help messages are attached to either the item or the window, the following message appears:

```
No help has been supplied for this topic
```

This default message is always the first message in every message file. You can modify the default message by using the Message Builder utility. However, you will first be warned as to the use of the default message. The default message cannot be deleted since Window Manager always expects message number 0 to be the default Help message.

When the Attach Help to Window option is selected at the window level, Help message numbers are attached to the window currently being edited. When the Attach Help option is selected at the item level, the cursor moves to the list of items and the help numbers are attached to the selected item. It should be noted that window help is only displayed when no item help is available. After selecting the Attach Help option at either level, you are requested to enter the pathname of the help file if a file has not been previously associated with this screen. The Attach Help Messages menu (Figure 4-4) then appears with all the commands, a list of the messages previously stored in the message file, and the list of messages previously attached to this item or window.

Figure 4-4

Attach Help Messages Menu

ATTACH HELP MESSAGES					
COMMANDS					
Add Help Message To List		View Help Message			
Insert Help Message Into List		Specify Help Message			
Delete Help Message From List					
ATTACHED HELP MESSAGES FOR WINDOW : 2 - Main Menu					
1	2				
HELP MESSAGES					
1	HELP MESSAGE1	2	HELP MESSAGE2	3	HELP MESSAGE3
4	HELP MESSAGE4	5	HELP MESSAGE5	6	HELP MESSAGE6
7	HELP MESSAGE7	8	HELP MESSAGE8		
Select a Command. Press the F10 key to return to the previous menu.					

If the message file is a new file with no previously saved Help messages, then Specify Help Message is the only option visible in the Command window. Once a Help message has been added to the file, the menu is repainted with all of the options visible. The following paragraphs describe these options.

When you have finished attaching help to a window or item, press the **F10** key to return to the window-level or item-level screen.

Add Help Message to List

4.6.1 This command is used to add another message to the list of messages attached to the window or item. The cursor will move into the list of Help messages. The message which is to be attached can then be selected and the number will appear in the list of attached Help messages. Pressing the **ESC** key before selecting a message aborts this procedure.

**Insert Help
Message Into List**

4.6.2 This option causes the cursor to move into the list of Help messages. Once the desired Help message is selected, the cursor moves into the ATTACHED HELP MESSAGES FOR WINDOW window. The Help message number from the Help messages list is inserted before the number selected from the ATTACHED HELP MESSAGES FOR WINDOW window. Pressing the **ESC** key at any time prior to selecting a position aborts this procedure.

**Delete Help
Message From List**

4.6.3 This option causes the cursor to move into the ATTACHED HELP MESSAGES FOR WINDOW window. The message number selected is then deleted from the list of attached messages. If there are no attached messages, the cursor does not move from the Delete Help option.

View Help Message

4.6.4 This option causes the Help message selected from the list of Help messages to be displayed on the screen. The message is placed in a window with a help label at the top. The window is positioned according to the coordinates specified for this Help message. The window is centered on the screen if coordinates have not been specified.

When you view a user-defined Help message, the message text is placed in a window drawn to the specified coordinates, but no special formatting of the message is done. See Chapter 8, User-Defined Windows, for more information on user-defined messages.

**Specify Help
Message**

4.6.5 This option permits a quick method of adding a new message to the message file without leaving Screen Builder and entering Message Builder. However, the capabilities of this option are much more limited than those provided by Message Builder. It is recommended that Message Builder be used for extensive specification and manipulation of messages.

After this option is selected, you will be presented with two pop-up windows in which to specify the message text and message name only. The message type will be forced to be a Help message, and the window will be centered on the screen with a window size dependent on the size of the message.

NOTE: Because of the way Screen Builder stores your help messages, it is necessary to have enough room on the default disk drive for the utility to store a temporary copy of your message file. If you receive an Error message referencing **MMTEMP. \$\$\$** or **MMTEMP2. \$\$\$** while your messages are being stored, you will need to make more room on your default disk. It is recommended that this utility be run from a Winchester disk drive.

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
5.1	Using Message Builder	5-3
5.2	Message Builder Options	5-3
5.2.1	Add Message	5-5
5.2.2	Modify Message	5-6
5.2.3	Rename Message	5-6
5.2.4	Change Message Type/Class	5-7
5.2.5	Copy Message	5-7
5.2.6	Reuse Message	5-7
5.2.7	Specify Window Coordinates	5-7
5.2.8	Delete Message	5-8
5.2.9	View Message	5-8
5.2.10	List Messages	5-8
5.2.11	Quit	5-9
5.3	Default Message	5-9
5.4	Using Message Manager	5-9
5.4.1	Variable Text	5-10
5.4.2	Message Manager Errors	5-11

Using Message Builder

5.1 Message Builder is an interactive utility that aids the application designer in constructing messages for use in an application program. The messages can be of four different types:

- Help
- Error
- Warning
- Please Note

Each message is classified as either a text message or a user-defined message. Once constructed, these messages are stored in a file and can be selectively displayed by making calls to Message Manager, discussed later in this chapter. Also refer to Chapter 6, Window Manager Callable Routines.

The `MBUILD` command invokes the Message Builder utility. Before entering Message Builder, you should first use the MS-DOS `SET` command to set up the environment variable `NLXTOOLS`. `NLXTOOLS` must be set to the directory pathname in which the screen files and message files for `MBUILD` exist (`.PIC`, `.NS$`, and `.NM$`). If it is not, `MBUILD` expects these support files to be on the default drive and directory. If you are using a pre-2.0 version of the operating system that does not support either directories or the `SET` command, ignore `NLXTOOLS` and ensure that the Message Builder support files are on the default drive.

Message Builder Options

5.2 Message Builder allows you to create and save messages in a format that can be easily accessed by Message Manager. Therefore, in order for this utility to execute, it must have a message pathname. There are two ways to specify a message file for Message Builder:

- The most efficient way is to include the file pathname on the command line after typing `MBUILD`. This allows an existing message file to be loaded along with Message Builder or a new file to be automatically initialized before the main menu appears.
- Another way is to type `MBUILD`, press the **ENTER** key, and wait for the system to prompt for a message file pathname.

Once a valid filename has been entered, the Message Builder menu (Figure 5-1) appears. If you have specified an existing message file, the messages contained in that file are listed in the lower half of the menu under the command options. If the file you have specified does not exist, Message Builder creates a default message (message number 0) to initialize the new message file and lists the message number for you at the start of the utility.

Figure 5-1

Message Builder Utility

NATURALLINK MESSAGE BUILDER UTILITY				
Commands				
	Add Message			Specify Window Coordinates
	Modify Message			Delete Message
	Rename Message			View Message
	Change Message Type/Class			List Messages
	Copy Message			Quit
	Reuse Message 2			
Messages from msgfile.txt				
0	Default Msg	1	First Message	3 Third Message
4	Fourth Message	5	Fifth Message	
Select a Command				

The command options available in Message Builder depend on the status of the message file specified, as well as the changes made to it. If the message file is a new file with no previously saved messages, all the options except the Delete Message and the Reuse Message options are displayed. Only after a second message has been added to a new file does the Delete Message option appear. This is due to the fact that the default message can never be deleted (as discussed in paragraph 5.3, Default Message). For either a new or an existing message file, the Reuse Message option becomes visible only after a message has been deleted, and remains visible as long as there are numbers from deleted messages which have not been reused.

Add Message 5.2.1 Selecting this option results in an edit window appearing, into which you can enter the text of the message. Special editing features have been added to this window and are documented with Help messages attached to the window. The editing keys include the following:

- **INS** — Enables you to insert a character or a string of characters.
- **DEL** — Deletes a single character.
- **CTRL-F1** — Moves the cursor to the beginning of a line of text.
- **CTRL-F2** — Moves the cursor to the end of a line of text.
- **CTRL-F3** — Moves the cursor left one word at a time.
- **CTRL-F4** — Moves the cursor right one word at a time.
- **CTRL-F5** — Deletes text from the cursor position to the end of the line. If the cursor is at the first character position, **CTRL-F5** removes the entire line from the window.
- **CTRL-F6** — Inserts a new line into the message text. A new line cannot be inserted if the last line in the window contains text. There is a limit of 19 lines per message.
- **F5** — Joins the line below the cursor with the line containing the cursor.
- **F6** — Breaks the line at the cursor and places the remainder onto a new line below the line containing the cursor. Since there is a limit of 19 lines of text for a Help message, the break operation cannot be performed if line 19 has text in it.

Each line of text is centered in the help window. To left justify a message, you must pad each line with spaces. If you want blank lines between lines of text, skip those lines as you enter the text. Press the **F10** key to save the text. Once the text has been entered, the system uses a second pop-up edit window to prompt for a name to identify the new message.

After you enter the message name, a third pop-up window allows you to select the message type and to classify the message as a text message or a user-defined message. The type of message selected determines the window label that will appear in the message window. The message classification tells Message Manager what to do with the message (refer to paragraph 5.4, Using Message Manager). If you classify the message as user-defined, the system automatically pops up the window used by the Specify Window Coordinates option, with the coordinates defaulted to a full-screen window. You can modify the coordinates prior to pressing the **F10** key to save them. The number and name of the new message are appended to the list of messages displayed under the command options on the Message Builder menu.

Press the **ESC** key at any time to abort the Add Message procedure.

Modify Message **5.2.2** Selecting this option causes the cursor to be placed in the list of messages. Once the message to be modified has been selected, the edit window used by the Add Message option appears containing the current message text. This text can then be modified.

There is one exception to the process just described. If you select the default message (message 0), the system intervenes with a message informing you of the special significance of the default message. (Refer to paragraph 5.3, Default Message.) After viewing this message, press the **ENTER** key and the current message text appears. Pressing the **F10** key stores any new changes made to the text. Pressing the **ESC** key aborts the Modify procedure.

Rename Message **5.2.3** When Rename Message is selected, the cursor is placed on the list of messages. Once the message to be renamed has been selected, a pop-up window containing the message name appears. The name can be changed or modified.

The processing exception described in the Modify Message option applies here also. When you select the default message (message 0) to be renamed, the system intervenes with an informative message prior to displaying the current message name. Pressing the **ESC** key aborts this procedure.

**Change
Message
Type/Class**

5.2.4 Selecting this option causes the cursor to be placed on the list of message names. Once the desired message has been selected, the pop-up window used by the Add Message option appears. The current message type and classification are highlighted in reverse video. These attributes can then be modified. Pressing the **ESC** key aborts this procedure.

Copy Message

5.2.5 This option enables a previously specified message to be copied to another previously added message. When the option is selected, the cursor is positioned in the list of message names. The name of the master message should be selected first, then the cursor should be moved to the name of the destination message. When the second selection is made, the text, type, classification, and coordinates of the master message are assigned to the destination message. Only the name of the destination message remains the same. Pressing the **ESC** key before selecting the destination message aborts this procedure.

Reuse Message

5.2.6 This option is visible only after a message has been deleted from the list. When the option is selected, the result is the same as the Add Message option. You are given a pop-up window into which text can be entered and then saved by pressing the **F10** key. You are prompted for a message name, type, classification, and, if the user-defined classification is selected, window coordinates.

The system keeps track of the numbers which are available to be reused and, when no more exist, it removes this choice from the list of options. When this option is available, you are advised to use it rather than the Add Message option to create a new message. Because of the way the message file is stored, this procedure helps minimize the size of the message file.

**Specify Window
Coordinates**

5.2.7 Selecting this option causes the cursor to be placed in the list of messages. From this list, you should select the message requiring specific coordinates. A pop-up window then prompts you for the upper left and lower right coordinates of the window in which the message will be displayed. If coordinates are not specified for a message, the window is shrunk to fit the message text and placed in the center of the screen. However, if the message you selected from the list is classified as user-defined, coordinates of all zeros will not be accepted.

To put a message into a window, the interior of the window must be at least three lines long and six columns wide. When specifying coordinates for the window border, it is necessary that the bottom row coordinate be at least four units larger than the top row coordinate and that the right-most column coordinate be at least seven units larger than the left-most column coordinates. Additionally, text that does not fit into the specified window is wrapped within the window, using the carriage return to indicate a new line. Pressing the **ESC** key aborts this operation.

Delete Message **5.2.8** Selecting this option causes the cursor to be positioned at the list of message names. The message selected is deleted from the list of messages. Since the default message (message 0) should never be deleted, it is made unselectable and the cursor is not permitted to move to it. Pressing the **ESC** key before selecting a message aborts this procedure. Once a message has been deleted, the Reuse Message command will become available and can be selected to reuse the message number for another message.

View Message **5.2.9** Selecting this option causes a message selected from the list of message names to be displayed on the screen. For text messages, the message is placed in a window with an appropriate label as if the message had been displayed by Message Manager, except that any variable text (designated by @ followed by an integer — that is, @1) in the message is not replaced.

When you view a user-defined message, the message text entered is placed in a window drawn to the specified coordinates, but no special formatting of the message is done.

List Messages **5.2.10** By selecting the List Messages option, you can have a formatted copy of the messages sent to a file. The listing can then be printed. This listing file should not be confused with the file in which the messages are stored for use by Message Manager. A pop-up window requests the pathname of the file that is to receive the list of messages. The messages are placed in the file in the following format:

```
1 MESSAGE NAME (MESSAGE TYPE)
      [ULX=0, ULY=0, LRX=0, LRY=0]
  MESSAGE TEXT
```

Pressing the **ESC** key before entering the pathname aborts this procedure.

Quit 5.2.11 Selecting this option terminates Message Builder. All new messages are stored in the file specified when entering Message Builder.

NOTE: Because of the way the Message Builder utility stores your messages, the default disk drive must have enough room for the utility to store a temporary copy of your message file. If you receive an error message referencing MMTEMP.\$\$\$ or MMTEMP2.\$\$\$ while your messages are being stored, you need to make more room on your default disk. It is recommended that this utility be run from a Winchester disk drive.

Default Message

5.3 Message files created with Message Builder or Screen Builder contain a default message (message 0). It is displayed when the Help key is pressed and no help has been attached to the item the cursor is on or to the window. It is also displayed when display of message number 0 is requested. The default message reads as follows:

No help has been supplied for this topic

You can modify the default message by using MBUILD. However, you will first be warned as to the use of the default message. The default message cannot be deleted since Window Manager always expects message number 0 to be the default Help message.

Using Message Manager

5.4 When the messages have been created and stored in a file, they can be accessed from the application program. The application program uses the Message Manager call to display the desired message during run time. Message Manager reads the binary file created by Message Builder and internally constructs a temporary window using the coordinates provided.

If no coordinates are specified (a situation that can exist only for text messages), Message Manager shrinks the window to fit around the message and positions it to appear in the middle of the screen when it is subsequently displayed. However, prior to displaying any message, Message Manager checks the message classification to determine the action to take next. If it is a text message, Message Manager displays it and waits for you to press the **ENTER** key. After receiving this signal to continue, Message Manager erases the message and returns to the calling procedure. The windows covered by the message window will be flagged for repainting or optionally redisplayed by Message Manager before returning.

Message Manager does not display user-defined messages. Instead, a call is made to an application routine, provided by you, to handle the message. Like Message Manager handling a text message, your application assumes responsibility for displaying the desired information and for deleting it. All Window Manager calls are available to you from your application. Refer to Chapter 8, User-Defined Windows, for a detailed discussion of the call to the application.

If the requested message cannot be found in the message file, the following appears:

```
An invalid message number has been supplied to message manager
The invalid number is: @1
```

The @1 will be replaced by the number of the requested message.

Variable Text **5.4.1** Messages constructed with Message Builder can contain variable text. That is, there may be slots in the messages where other strings are to be inserted. These slots are indicated by an at sign (@) followed by an integer. For example, @1 indicates that the first string is to be inserted here, @2 indicates that the second string is to be inserted here, and so on. These can be repeated, so that the same string can be inserted into several slots. Thus, a valid message text may be:

```
@1 @1 little @2
```

In the preceding example, if the string *twinkle* is the first string to be inserted, and *star* the second, the message displayed by Message Manager is as follows:

```
twinkle twinkle little star
```

The message appears in a window that can either be centered on the screen or placed according to specified coordinates.

**Message
Manager
Errors**

5.4.2 An unformatted Error message appears if Message Manager cannot correctly display a message. This message is as follows:

```
An error with the following code was encountered: XXX...
The message display system was unable to neatly process
this message for the following reason: XXX...
```

Following is a list of the error codes encountered and explanations of each:

- Message files read error — This means that the message has been damaged or deleted, or that the filename passed to Message Manager was not a message file.
- Couldn't get memory for messages — This indicates that there was not enough memory to display the message.
- Invalid filename passed to Message Manager — This indicates that the name of the message file passed to Message Manager does not follow correct operating system naming conventions.

WINDOW MANAGER CALLABLE ROUTINES

6

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
6.1	Procedure Calls.....	6-3
6.1.1	Initialize Window Manager.....	6-3
6.1.2	Load Screen File.....	6-4
6.1.3	Add Window.....	6-4
6.1.4	Select Window.....	6-4
6.1.5	Refresh Window.....	6-5
6.1.6	Display Window.....	6-5
6.1.7	Receive From Window.....	6-5
6.1.8	Release Window.....	6-6
6.1.9	Delete Window.....	6-7
6.1.10	Save Screen.....	6-7
6.1.11	Unload Screen.....	6-7
6.1.12	Reset Window Manager.....	6-7
6.1.13	Add Item.....	6-7
6.1.14	Insert Item.....	6-8
6.1.15	Delete Item.....	6-8
6.1.16	Create Item Table.....	6-8
6.1.17	Create Window.....	6-9
6.1.18	Get Attribute Value.....	6-9
6.1.19	Get String Value.....	6-9
6.1.20	Set Attribute Value.....	6-9
6.1.21	Set String Value.....	6-9
6.1.22	Clear Screen.....	6-9
6.1.23	Display Message From Message Manager.....	6-10
6.1.24	Reset NaturalLink Memory.....	6-10
6.2	Typical Calling Sequence.....	6-11

Procedure Calls

6.1 Window Manager controls windows and associated text items and can perform various functions on them. However, Window Manager must be called upon by an application program in order to perform these functions. Since the windows used by application programs are stored as screen data files, each application program using Window Manager must keep track of where and how the windows are to be used in the program. The application program must then make certain procedure calls to Window Manager to invoke the functions. The nature of these procedure calls is covered in this chapter. Tables listing the actual calls an application program uses are presented in the appendixes.

An application program calls Window Manager through a high-level language interface. How the calls are made depends on the programming language used. An explanation of what kinds of calls are made is summarized here.

Initialize Window Manager

6.1.1 Window Manager must know what hardware the application is running on. The Initialize routine does this machine detection and sets the flag Llpctype (pctype for FORTRAN) accordingly. If the internal phrase file (NLXPHRAS.NM\$) is present, it will be loaded during this initialization time. (See Chapter 9, Internal Phrase Editing, for more information.)

Window Manager also requires that certain cursor and video display attributes be set before it starts processing; on some machines it invokes its own keyboard mapping if necessary. (See Appendix C, Computer-Specific Information, for more information.) To perform these operations, your application program **must** call the Initialize procedure before calling any other Window Manager routines. This routine only has to be called once before any other Window Manager routines are called.

If the Initialize procedure is not called, Window Manager will not run properly.

Load Screen File **6.1.2** Window descriptions are stored in files built by Screen Builder. These window descriptions must be entered into memory before Window Manager can perform any other functions on them. The Load procedure reads the screen file, allocates memory for each stored window description, and loads the window descriptions into memory. A maximum of 20 screen files can be loaded into memory at any one point.

Add Window **6.1.3** After all windows are loaded into memory, they must be added before any Select, Receive, Refresh, Release, or Delete function can be performed on them. The Add call makes a window known to Window Manager. An added window is then flagged to have its text and border painted. This procedure only adds the window; the window is not displayed until a Receive call is made for any window on the screen or until a Receive or Display call is specifically made for this particular window. If the window was already added, this procedure returns a warning code and the Add procedure is not performed.

The file text for a file window is loaded at the time the Add call is made. If for some reason the file window text cannot be loaded, the Add call returns an error code, but the window is still added. In other words, the window is added even though it has no items.

Once a window has been added, Window Manager remains aware of it until it is deleted. Since adding a window flags the window to be painted, windows should not be added until they are to become visible. Once the window has been displayed (by either a Display call or a Receive call), it remains showing unless it becomes covered by another window or is deleted. If the window is covered by another window, the covered window is still known to Window Manager and will be flagged for repainting when the covering window is deleted.

Select Window **6.1.4** This procedure activates a window. It also flags the window to have its text repainted. The window is repainted the next time a Receive call is performed on any window. If the window was not added or if the window is already active, this procedure returns a warning code and the Select procedure is not performed.

Refresh Window

6.1.5 This procedure flags a window to have its text and borders repainted. The window is redisplayed the next time a Receive call is made. This procedure is used whenever the application has changed the window and the change needs to be shown. If the window was not added or if the window is already flagged to be repainted, this procedure returns a warning code and the Refresh procedure is not performed.

At the time the Refresh call is made, the file text for a file window is reloaded. If for some reason the file window text cannot be loaded, the Refresh call returns an error code, but the window is still refreshed. In other words, the window no longer has any items in it but will still appear on the display device.

Display Window

6.1.6 This procedure repaints the text and border of the window without any delay. This procedure is used whenever a window needs to be displayed, but having the display triggered by a Receive call does not work. It follows the same procedure as Refresh, but there is no delay in the repainting process and the text for a file window is not reloaded. If the window was not added, this procedure returns a warning and the Display operation is not performed.

Once a window has been displayed (either by a Display call or by a Receive call), it continues to be displayed unless it becomes covered by another window or is deleted. If the window is covered by another window, the covered window is still known to Window Manager and will be flagged for repainting when the covering window is deleted.

Receive From Window

6.1.7 This procedure receives information from any currently active window. The Receive procedure requires three main parameters.

- **Window** — This is the window from which input is received. Window Manager places the cursor in this window first. However, if this is not a valid window from which to receive information, Window Manager places the cursor in the first active window it finds among all the windows currently added. A valid window is an active, nondisplay-type window that has at least one selectable item in it. If there are no active windows or if no windows at all are available, an error code is returned by the procedure.

On return from the Receive call, this parameter generally contains the window from which a selection has just been made. However, if there are two or more active or inactive windows on the display, this parameter can contain any one of them. A user always has the opportunity to move to a different window if the appropriate keys are defined. Refer to Chapter 10, Window Manager Input Devices, for further details on changing the function keys.

If there are two active windows on the display, the user can move the cursor to the other active window and make a selection. The window from which the selection is made is returned to the application. A more complicated case can occur. If a user moves to an inactive window and then presses a key undefined to Window Manager, this inactive window is returned. Therefore, it is recommended that the application program check the returned window to ensure that it is the window expected by the application program.

- **ITEM** — This is the item index. Window Manager uses this field as the starting item for the cursor. If this field is out of the range of the items or if the item referenced is invisible or unselectable, Window Manager places the cursor on the item it was last on in that window. If the cursor has never been in the window before, the first item in the window is used. On return from Receive, this field contains the value of the item the cursor was on; depending on the key pressed, this is usually the selected item.
- **KEY** — This parameter contains a return value. It is the value of the key that, when pressed, causes Window Manager to return.

Receive is the most powerful call of all Window Manager routines. If the window specified in the Receive call does not have the Special Repaint on Receive attribute set to 1, all windows flagged to be repainted are repainted at this time. Redisplaying the screen only when a Receive call is made eliminates excessive screen flashing caused in the repainting. It is important to note that just specifying a window in a Receive call is not enough to cause it to be repainted. The window must be flagged to be repainted by one of the other Window Manager calls before the Receive will trigger its repainting. When a Receive call is made, Window Manager takes complete control of the interaction until a key is pressed which has not been assigned to a Window Manager function.

The text in an edit window is padded with fill characters before a Receive call is made and is stripped of these fill characters after the Receive. The application program does not have to perform this function.

Release Window **6.1.8** A call to this procedure makes a window inactive and flags the window to have only its text repainted. If the window was not added, or if it is already inactive, this procedure returns a warning code and the release operation is not performed.

Delete Window **6.1.9** This procedure deletes a window from the screen and makes it unknown to Window Manager. The deleted window is also made inactive. A scan is performed to determine if the deleted window was covering any other window(s) on the screen. If so, these windows are flagged to have both their borders and text repainted. If the window was not added or if the window was already deleted, this procedure returns a warning code and the Delete procedure is not performed.

Save Screen **6.1.10** This procedure saves a screen to a file. It does not unload the screen from memory, and any Window Manager procedure can be performed on the windows in this screen after the Save procedure has been performed. Screens do not have to be saved to their original files. However, if the screen is saved to the original file, the original will be overwritten. This procedure is useful if windows in a screen have been modified and these modifications need to be saved for future use.

Unload Screen **6.1.11** This procedure releases the memory used by Window Manager to store the window descriptions of the screen file. The Unload procedure does not save the window descriptions. If the windows in the screen being unloaded have not been deleted yet, the Unload procedure automatically deletes them.

Reset Window Manager **6.1.12** This procedure restores the cursor and video display attributes and the keyboard mapping values to their original state. The function it performs is opposite to the function that the Initialize procedure performs. Reset **must** be called before the application program terminates. It should be the last Window Manager call made. If this call is not made, Window Manager does not terminate correctly, causing the cursor and video display attributes to be incorrect. Also, since the keyboard mapping (performed by Initialize) will not be restored without this procedure, a system reinitialization will be required.

Add Item **6.1.13** This procedure adds a new item to the end of the list of items for a given window. The item added has the default attributes of visible, selectable, intensity level of 7, reverse video, required input, echo input, and item label intensity of 7. The default value of all other item attributes is 0.

Insert Item **6.1.14** This procedure inserts an item at a specified place into the list of items for a given window. The attributes of the inserted item take on, as default values, the same values as in the Add Item procedure. After insertion, all items following the inserted item are renumbered.

Delete Item **6.1.15** This procedure deletes a specified item from a given window. All items following the deleted item are renumbered.

Create Item Table **6.1.16** This procedure creates an item table of a given size for a given window. This is a very powerful call for item table manipulation because it can be used in four different ways.

- If no items currently exist for the window, this procedure creates a specific number of items. The values of their attributes take on, as default values, the attributes used in the Add Item procedure.
- If the number of items to be created is greater than the current number of items for the window, the item table is increased in size by the difference between the two values. This is the equivalent of making several Add Item calls.
- If the number of items to be created is smaller than the current number of items for the window, the item table is decreased in size by the difference between the two numbers. This is equivalent to several Delete Item calls on the last item in the list.
- If the number of items to be created is 0, all of the items in the item table are deleted.

The Create Item routine will return a warning code if any items are deleted to make sure the application knows what is happening. The Create Item routine is more efficient than separate Add or Delete Window calls. It should be used if possible when creating or deleting several items.

Create Window **6.1.17** With the Create Window call, you can add a window to an existing screen or create a new window in a new screen. The Create Window call takes both a screen and a window number as parameters. The screen is used both for input and output, while the window is used only for output. If the screen passed is currently loaded, a new window will be added to the existing screen and the newly created window number will be passed back to the application. If the screen does not exist, a new screen and window will be created and the new values for both will be passed back.

The created window will have the same default attributes as a newly added window in Screen Builder. Once you have created the window, all the other Window Manager calls can be made using the new window. You can create items for it, change its attributes, add it, select it, and receive information from it.

Get Attribute Value **6.1.18** This procedure gets the value of a numeric attribute for a given window or item.

Get String Value **6.1.19** This procedure gets the value of a string attribute for a given window or item. If the item text for an edit window is retrieved, Get String Value will strip off the fill characters before returning the string value.

Set Attribute Value **6.1.20** This procedure allows you to assign a value to a numeric attribute for a given window or item. A Refresh call followed by a Receive call or a Display call must be made to make Window Manager aware of the change.

Set String Value **6.1.21** This procedure allows you to assign a string value to a string attribute for a given window or item. A Refresh call followed by a Receive call or a Display call must be made to make Window Manager aware of the change.

Clear Screen **6.1.22** This procedure clears the display area of any text currently displayed. This can be old text or even a window displayed by Window Manager. This procedure only clears the display area itself; it is **not** the same as making any of the other Window Manager procedure calls. Windows removed from the display by the Clear Screen procedure are **not** deleted. To redisplay them, the application program must make either a Display or Refresh call followed by a Receive call. This call deletes only the text from the screen. It will **not** clear any graphics.

**Display Message
From Message
Manager**

6.1.23 This procedure displays a given message created previously with the NaturalLink Message Builder utility. If the message contains slots for variable text, the text to be inserted must also be supplied. (See Chapter 5, Message Builder, for more information about using Message Manager.)

**Reset
NaturalLink
Memory**

6.1.24 The Window Manager and Toolkit (if you are using it) run-time use a data area separate from your application program. During some applications, the NaturalLink memory area may become fragmented from the loading and unloading of screens or from other operations which result in memory usage. This could ultimately result in a Get Memory error, where the NaturalLink run time cannot get the memory it needs. When a memory error occurs, the NaturalLink data area will be cleared and reset. This way the application can reload screens and other data, and processing can continue.

For some applications, just allowing the NaturalLink run-time to run out of memory without notice may cause an inconvenient delay while the application reloads screens and processing resumes. In cases like these, the Reset Memory call can be used to force a cleanup of the NaturalLink data area at a more opportune time. The data area will be erased and reset to the way it was before the application had been run. No screens or other data will be saved; this must be done by the application before making the Reset Memory call if it is desired. Once this call is made, any screens which were loaded into memory are cleared and must be reloaded before the windows in them can be accessed again. Remember, this call affects only the NaturalLink data space; anything stored in the application data area will not be affected.

Typical Calling Sequence

6.2 A typical calling sequence is as follows:

1. Initialize Window Manager — Detects the machine on which the program is running, sets cursor and video display attributes, and invokes keyboard mapping (if used).
2. Load Screen File — Loads windows into memory.
3. Add Window — Adds the window, making it known to Window Manager.
4. Select Window — Makes the window active and ready for a Receive call.
5. Receive From Window — Gets information from an active window.
6. Release Window — Makes the window inactive.
7. Delete Window — Deletes the window.
8. Unload Screen — Releases the memory taken up by window descriptions.
9. Reset Window Manager — Restores cursor and video display attributes and any keyboard mapping.

APPLICATION VALIDATION ROUTINE

7

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
7.1	Definition	7-3
7.2	Uses for the Routine	7-3
7.3	When Validation Is Done	7-4
7.4	How It Works.....	7-4
7.5	Parameters and Return Status	7-5
7.5.1	Datatype Parameter.....	7-5
7.5.2	Return Status for Validation Process	7-5
7.5.3	Item Number Parameter	7-6
7.5.4	Typical Algorithm for Validation Code	7-7
7.6	Restrictions on Use of Validation Routine	7-8
7.7	Dummy Validation Routine	7-8

Definition

7.1 Chapter 6, Window Manager Callable Routines, discusses all the calls that the application program can make to Window Manager. This chapter and Chapter 8, User-Defined Windows, discuss calls that Window Manager makes to the application program. These calls allow the developer to make the application more flexible.

The Application Validation routine provides a way to perform data checking on information entered in edit windows. Datatypes may be set for each item in the window, and the validation routine is called to do type-checking based on this datatype. The validation routine, which must be supplied by the application developer, is called when the cursor moves off the current edit field. This can occur when the user presses the **ENTER** key or any of the normal cursor movement keys.

Uses for the Routine

7.2 In some Window Manager applications, specific information must be obtained from the user. This is accomplished by using an edit window. The data the user enters must be checked to ensure that a valid entry was made. Since Window Manager does not perform data checks, this check must be done by the application when control is returned to the application program.

A problem occurs in doing validation on the data entered for any edit item. Single-selection edit items are the only fields that can be type-checked immediately following data entry. However, the user cannot move to other fields in the window to change their values. Multiple-selection edit windows allow the user to move around and change all fields, but the application is unable to type-check anything until all editing is complete. This arrangement does not provide all the functionality that is needed in certain edit situations. Specifically, this causes problems in a data entry application when a user must be able to change a previously entered field, but the changes may affect other fields and must be type-checked immediately.

This is where the validation routine is applied. Since the validation routine is called each time the user completes or moves off an edit field, the application can easily check the data before the user is allowed to enter information for another field. If the data entered is incorrect, the validation routine can display an error message and force the user to try again. If the data entered has a direct effect on a previously entered field, the validation routine can tell Window Manager to put the cursor on that previous field so the user can reenter the data for that field.

When Validation Is Done

7.3 The validation routine is called when a key is pressed that would move the cursor off the current item to a new item. The new item may or may not be in the same window. The following keys are included:

1. Select and Proceed — **ENTER** and **F10**
2. Next item — **Up** and **down arrow**, **TAB** and **SHIFT-TAB**
3. Page Scrolling — **F1** and **F2** (can be used only in multiple-selection windows)
4. Top/Bottom Item — **HOME** (can be used only in multiple-selection windows)
5. Next Active Window — **CTRL-HOME**
6. Next Window — **CTRL-up** and **down arrows**, **CTRL-PgUp** and **PgDn**
7. Any Unknown Key — Keys unknown to Window Manager that would make Window Manager return to the application program.

How It Works

7.4 Validation is performed by taking the following steps:

1. When the window is specified in Screen Builder, a datatype is set for each item.
2. The developer writes the Window Manager application and also writes a Validation routine to check the item for correct data based on the datatype.
3. The application is then linked with the Window Manager run time.
4. The application is run, and a Receive call is made on the edit window. When the cursor is moved off the current edit field, either by a cursor movement key or by committal of the edit field with the **ENTER** or the **F10** key, the validation routine is called.
5. The validation routine checks the data entered, based on the datatype of the item. When the check is complete, the validation routine returns to Window Manager with the status of the validation.
6. Window Manager proceeds based on the status code returned and the key pressed.

Parameters and Return Status

7.5 The validation routine has the following parameters. (For the exact calling sequence, refer to the appropriate appendix for the language you are using.)

```
APPVAL(SCREEN#, WINDOW#, ITEM#, KEY, DATATYPE)
```

For the most part, these parameters are self-explanatory. `SCREEN#`, `WINDOW#`, and `ITEM#` indicate the specific item being validated. The `KEY` parameter indicates the key pressed to initiate the validation, and `DATATYPE` is the datatype assigned to this item.

Datatype Parameter

7.5.1 The datatype attribute is used to indicate to the validation routine the kind of information that can be entered for the item. The valid datatypes are 0 through 200. The datatype can be assigned either in Screen Builder or in the application program.

A datatype of 0 indicates that Window Manager is not to call `APPVAL` for this item; the key pressed is processed as usual. This is useful if you do not care what is entered for the item or if you want to check all the items at one time (when Window Manager normally returns to the application).

The rest of the datatypes assigned to the items have meaning only to `APPVAL`. Since only one validation routine is used for an application, if the same datatype is assigned to more than one item (even if the items are in different screens), the validation performed is the same for both items. For example, if you wish to perform filename validation for several different items, assign the same datatype number to each of the items. When you detect this datatype in your validation routine, you can perform the same filename validation regardless of the item or window the input came from.

Return Status for Validation Process

7.5.2 `APPVAL` must return a status code to Window Manager indicating the results of the validation process. The status code must be one of the following:

1. `INVALID/CLEAR` — value `-2`. Type-check failed. Window Manager clears the text from the field being checked and puts the cursor at the start of the field so the user can try again. If any error message is desired, `APPVAL` is responsible for displaying and deleting the message.
2. `INVALID/NO CLEAR` — value `-1`. Type-check failed. Window Manager places the cursor at the start of the field so the user can try again. Any text entered prior to the validation call remains in the field. If any error message is desired, `APPVAL` is responsible for displaying and deleting the message.

3. VALID — value 0. Type-check passed. Window Manager performs the next functionality based on the key pressed and the selection mode — such as move to another edit field or exit the Receive call.
4. IGNORE — value 1. Type-check not done. This code is used when the validation routine did not bother to check the entered data. The application may want to use this when APPVAL is called because an unknown key is pressed (that is, a key that Window Manager does not handle and that would normally cause Window Manager to return to the application). Window Manager leaves the cursor where it was so the user can continue typing text. When the call returns to Window Manager, Window Manager does *not* return to the application even if the check was generated by an unknown key or any key which normally causes Window Manager to return to the application.
5. EXIT — value 2. Type-check may or may not have passed. APPVAL returns this code when it wants Window Manager to exit the Receive call immediately. Window Manager will then return to the application program with a special warning status to indicate what happened.

If a code other than one listed here is returned, a valid status is assumed by Window Manager.

Item Number **7.5.3** One of the parameters for the validation call is the item number of
Parameter the item on which the validation needs to be done. The validation routine
 needs this number to get the item text entered by the user.

The item parameter is also used by Window Manager when the validation routine returns to Window Manager. Window Manager always puts the cursor on the item specified by the item parameter if the item parameter has been modified by the validation routine. Window Manager does this regardless of which key was pressed.

The exception to this rule occurs if a status of EXIT(2) is returned. If the exit status is returned, Window Manager returns to the application regardless of which key was pressed.

The following illustrates the way Window Manager handles the return from the validation routine.

```
case RETURN STATUS of
  INVALID:  if INVALID and CLEAR edit field status,
            clear out current edit field
            if item parameter is different than one passed in,
            find position of new item
            place cursor on that item
            else
            put cursor at start of the current item
            go get another key
  VALID:    if item parameter is different than one passed in,
            find position of new item
            place cursor on that item
            go get another key
            else
            handle key that caused validation call
  IGNORE:   if item parameter is different than one passed in,
            find position of new item
            place cursor on that item
            go get another key
  EXIT:     return to application.
```

Typical Algorithm for Validation Code

7.5.4 Following is a typical algorithm for the validation code.

```
handle any application-specific keys.
if the key is not one we want to deal with,
  return(ignore).
get the entered data - Window Manager get string call.
case datatype of
  1: do validation;
    if error,
      display error message.
      return(invalid/clear or invalid/no clear).
    if correct,
      return(valid).
  2: do validation
    .
    .
    .
```

Restrictions on Use of Validation Routine

7.6 Almost any Window Manager call can be made from the Validation routine. Windows can be added, displayed, received from, and deleted. Message Manager calls can be made to display error messages. Following are the only restrictions:

- A Receive call cannot be made on the same window that the validated item is in.
- The position of the validated item cannot be changed. This includes both the physical position on the screen and the item position/number.
- The maximum edit field length cannot be changed.
- Any windows displayed by the validation routine must be deleted by the validation routine or by the application program when the receive on the edit window is terminated. When the validation routine returns to Window Manager, Window Manager assumes the user has an uncluttered view of the edit window and its items.

Dummy Validation Routine

7.7 A dummy validation routine has been provided with the Window Manager run-time object code to resolve the reference to the validation call. This dummy routine can be linked with the application if no validation routine will ever be needed by the application, such as if all datatypes are 0. The dummy routine is shipped as part of a library of dummy routines. Consult the appropriate appendix for the language you are using for details.

The dummy routine always returns a status of VALID (0), even if an item has a datatype greater than 0.

USER-DEFINED WINDOWS

8

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
8.1	Introduction.....	8-3
8.2	Functionality	8-3
8.2.1	Specifying a UDW	8-3
8.2.2	How UDWs Work with Window Manager Run Time.....	8-4
8.2.2.1	Receive Call.....	8-4
8.2.2.2	Display Call	8-6
8.2.2.3	Delete Call	8-6
8.2.2.4	Other WM calls.....	8-7
8.3	User-Defined Messages	8-7
8.3.1	Specifying the Message	8-7
8.3.2	How the UDM Works	8-7
8.4	Uses for UDWs/UDMs.....	8-8
8.4.1	Graphic Title Window	8-8
8.4.2	Command Window	8-8
8.4.3	Selection From Icons.....	8-8
8.4.4	Graphical Help Messages	8-8
8.5	UDW/UDM Application Calls	8-9
8.5.1	Application Receive Call	8-9
8.5.2	Application Display Call.....	8-10
8.5.3	Application Delete Call	8-10
8.5.4	Application Message Manager Call.....	8-11
8.6	Making Window Manager Calls Inside a UDW Routine	8-11

Introduction

8.1 This chapter describes another method of making your Window Manager application even more flexible and elaborate. Window Manager is a powerful tool for developing menus; it provides ready-made window types and attributes to support the windowing needs of most developers. The user-defined windows (UDWs) feature allows the developer to create special windows while keeping the overall look of the application consistent.

UDWs are specified in Screen Builder like any other type of window. The only difference is that the application is in full control of any display, receive, and delete functions performed on UDWs. This feature provides a means of displaying data to and receiving it from the user in ways that Window Manager does not. Examples are the display of graphics pictures and selection of graphics text, icons, and items from a file.

Functionality

8.2 The UDW works like any other window, with one major exception: Window Manager and the application share control of the UDW. The UDW is tracked and flagged for repainting by Window Manager when it is added, selected, refreshed, released, or covered by another window. If the user has moved the cursor into the UDW, the application is notified by Window Manager. Display of the UDW is performed by the application program.

With a UDM, the procedure is the same except that an application Message routine is called to handle the display of the message, wait for user response, and delete the message. Message Manager displays and deletes the message window border, and flags for repainting or optionally redisplay the underlying windows.

Specifying a UDW

8.2.1 UDWs are specified in Screen Builder the same as are other windows. When setting window format attributes, the application developer selects User-Defined for the window type. Screen Builder then prompts for a UDW code ranging from 10 to 99. (This code is actually stored as the window type.) Once marked as a UDW, the only attributes which affect the appearance or performance of the window at run time are:

- Pop-up window attribute
- Special Repaint on Receive
- Allow Cursor to Enter Window
- All border attributes except scroll markers

The other attributes are optional, depending on what the application plans to do with the window; they are not checked by Window Manager.

The UDW code has no intrinsic meaning to Window Manager. A window type number between 10 and 99 designates a UDW. The same code can be used for many different UDWs. The developer might, for example, assign the same UDW code to all windows that perform a particular function. All UDWs with a particular code can then be treated one way, and those with other codes can be handled another way.

The UDW cannot be tested inside Screen Builder because the application must provide support routines to handle it.

**How UDWs Work
With Window Manager
Run Time**

8.2.2 Window Manager deals with UDWs at four different points (two of which are in one call):

- Receive and display (in one call)
- Display (by itself)
- Delete

UDWs are loaded, added, selected, released, and refreshed the same as other windows types. Differences occur when a Receive, Display, or Delete call is made.

Receive Call

8.2.2.1 The Receive call is the application's method of getting a response from the user. However, this is not its only function. The Receive call also triggers the painting of any window which has been flagged for painting. The following actions flag a window to be painted:

- Add
- Select
- Release
- Refresh
- Delete (referring to a window covering other windows)

When a Receive call is made on a window that is not a UDW, Window Manager scans the list of added windows for all windows needing repainting and then draws them as usual. However, if a UDW has been flagged, Window Manager clears the text from the screen area where the window will be drawn. Window Manager draws only the window border (if there is one) and calls the application's UDW Display routine. The application draws the window in whatever manner it desires and then returns to the Receive call, which progresses as usual.

If a receive is being done on a valid UDW (one that has been added, selected, and does not have the Allow Cursor to Enter Window attribute set to Never), the display process is the same. When all flagged windows are displayed, Window Manager calls the application's Receive routine; the application handles any cursor movement, scrolling, and the other function keys. The only requirement is that the application Receive routine return a valid window number, item number, and key pressed. When these values are returned to the Window Manager Receive routine, the key code is checked. If the key returned is neither a window movement key nor the Help key, Window Manager returns to the application.

Window-to-Window Movement UDWs are affected by The Window Manager capability to move the cursor from window to window until the desired window is reached. When a next window function is performed and the cursor enters the UDW, the application Receive routine is called. This is done even if the cursor is passing through the UDW to get to another window. The application Receive routine must be able to detect a next window key (defaults are **CTRL** with **HOME** or with an **arrow** key) so it can return right away. The Allow Cursor to Enter Window attribute can be used to limit or eliminate the cursor's ability to enter a UDW.

Processing Help If the Help key is pressed for a UDW, the application Receive routine can handle it in one of two ways.

1. Call Message Manager with the proper help number and help file.
2. Return to Window Manager, passing to Window Manager the Help key and the item the cursor was on. Window Manager then displays the Help message for the item. If the item number returned is not a valid item or if there is no help attached to the item, the Help message attached to the window is displayed. When the user has finished viewing the help, Window Manager calls the application Receive routine again to continue processing the Receive call.

The following pseudocode shows what happens during a Window Manager Receive call.

```
WMWRCV(SCREEN#,WINDOW#,ITEM#,KEY,ITEMTEXT)
  loop through added windows.
  if window needs repainting,
    call display routine.
nxtrecv:
  if this is a user-defined window,
    call the application receive routine.
  if key returned is a next window key,
    calculate which window to move to.
    go to nxtrecv.
  if key returned is help key,
    if item exists and help is attached,
      display item level help.
    else
      display window level help.
    go to nxtrecv.
  else
    return to application.
else
  window manager handles receive.
  return to application.
```

Display Call 8.2.2.2 UDWs are affected by the Display call just as they are by the Receive call repaint process. During a Display call, Window Manager determines if the window is a UDW. If so, it calls the same application Display routine that is invoked by the Window Manager Receive call. Pseudocode for this display process is as follows:

```
WMWDIS(SCREEN#,WINDOW#)
  if bordered window,
    display the border.
  clear text area of window.
  if user-defined window,
    call application display routine.
  else
    display the window text.
```

Delete Call 8.2.2.3 Because the application displays the UDW, it must also delete it. When the Delete call is made, Window Manager deletes the border (if there is one) and the text area of the window, flags all windows covered by the UDW to be repainted, and then calls the application's Delete Window routine for any part of the window that it cannot delete. Window Manager deletes only text, not graphics. Pseudocode for the Window Manager Delete call follows.

```
WMWDEL(SCREEN#,WINDOW#)
  if bordered window,
    delete the border.
  delete the text area of the window.
  flag all windows covered by the window.
  if this is user-defined window,
    call application delete window routine.
```


Other WM Calls **8.2.2.4** UDWs are not treated differently for any of the other Window Manager calls. Add, Select, Release, and Refresh calls all perform their normal functions. All four calls flag the UDW for repainting. Select turns on the active flag in the UDW; release turns it off. If the UDW is to be displayed differently depending on whether it is active or inactive, the application Display routine must check the active flag before doing the display.

User-Defined Messages

8.3 Related to the UDW is the user-defined message (UDM). This feature allows the application to control the messages displayed when the user presses the Help key or when a call to Message Manager is made. Through a user-defined message, a picture or diagram can be displayed with a Help or Error message.

Specifying the Message

8.3.1 Using the Message Builder utility, specify a UDM the same as you would any other message. Once you select message type (Help, Error, Warning, or Please Note), Message Builder prompts for whether it is a UDM. If you select yes, you must specify the coordinates for the message.

Specification of message text is optional for UDM. If text is specified, it will be loaded into the message window before the application Message routine is called.

How the UDM Works

8.3.2 When the user presses the Help key or the application makes a Display Message call, Message Manager is invoked. For Help messages, Window Manager gets the Help message number and filename stored in the window data structure and calls Message Manager with these parameters. If the developer calls Message Manager directly, the call must specify the message number and message filename as parameters. Message Manager reads the requested message from the file into memory and creates a window with the message text displayed. When the user presses the ENTER key, the message is deleted from the screen and the underlying windows are redisplayed.

With a UDM, the procedure is the same except that an application Message routine is called to handle the display of the message, wait for user response, and delete the message. Message Manager displays and deletes the message window border, and flags for repainting or optionally redisplay the underlying windows.

Message Manager looks for different input from the user, based on message type. For Help messages, Message Manager looks for three keys to be pressed: the Select key to terminate help; the Backup key to get the previous Help message; and the Help key to get the next Help message. For all other messages, only the Select key is sought to terminate the message. If one of these keys is not returned to Message Manager from the application Message routine, it assumes the Select key was pressed. Pseudocode for the Message Manager routine follows.

```

MSGMGR(MSGNUM, VARTXT, MSGFILE)
  nextmsg:
    read in requested message.
    create window with message as its text.
    display message window border.
    if user-defined message,
      call application to display message.
    else
      display the message.
      if help message displayed,
        wait for select, backup, or help keys.
      else
        wait for select key.
    if the help key is pressed,
      get the next help message.
      goto nextmsg.
    if the backup key is pressed,
      get previous help message.
      goto nextmsg.
    delete message window.
    return.

```

Uses for UDWs/UDMs

8.4 There are several ways in which UDWs or UDMs can be used. Some suggestions are presented in the following paragraphs.

Graphic Title Window

8.4.1 In many Window Manager applications, a title window is displayed at the top of every screen. A UDW could be used here to display graphics (such as a logo). Also, the text that is usually displayed as the title could be displayed in a different font by using a UDW.

Command Window

8.4.2 One possible application is a Command window that consists of selectable icons to do various functions such as quit and backup. This window has to be active at all times so the user can always select an icon. Since the application never knows when the receive on the icon window will be called, you should use a UDW.

Selection From Icons

8.4.3 UDWs can be created to display icons and allow selection of these icons. This is useful in applications where words alone will not convey the ideas to your users. A good example is the use of a map to indicate where cities are and to allow a city to be selected from the map.

Graphical Help Messages

8.4.4 As in the above example, words do not always fully explain an idea. For some Help messages, providing a picture along with the descriptive text would be useful in getting an idea across to the user. UDMs can be used for just such a purpose.

UDW/UDM Application Calls

8.5 This chapter details Window Manager calls the application must support when using the UDW and UDM options. Any nonzero status returned from one of these calls will cause Window Manager to return to the application with the returned status.

Application Receive Call

8.5.1 The application Receive call is almost the same as the Window Manager Receive call. When Window Manager calls the application Receive routine, it identifies the UDW by passing the screen number, window number, and the UDW code. The application must pass back the item selected, the key pressed, and the X and Y coordinates of the cursor.

The X and Y coordinates are used by Window Manager only if the key returned is a window-to-window movement key. The window that the cursor is moved into is based on the current position of the cursor. These coordinates must be character coordinates, with the X coordinate ranging from 0 to 79, and the Y coordinate ranging from 0 to 24.

Pseudocode for an application Receive routine follows.

```
APPRCV(SCREEN#,WINDOW#,ITEM#,KEY,UDWNBR, XPOS, YPOS)
  place the cursor on the initial item.
  nextinput:
    get user input.
    case input of
      cursor movement:
        move the cursor.
        go to next input.
      item selection:
        set item param = item selected.
        set key param = key pressed.
        return(0).
      window movement:
        set key param = key pressed.
        set X,Y param = cursor coordinates.
        return(0).
      help key:
        set key param = key pressed.
        return(0).
```

**Application
Display Call**

8.5.2 The application Display call is based on the window number or the UDW code. Both numbers are passed to the application, along with the screen number. There are two methods of handling this.

1. The developer can assign the same code to all UDWs displayed the same way. That code, which determines how the window will be displayed, is checked in the Display routine.
2. The application Display routine uses the window number to determine how the display is to be done.

The following Display call illustrates the first method.

```
APPDIS(SCREEN#,WINDOW#,UDWNBR)
  get the window coordinates.
  case UDWNBR of
    10: display window contents.
    20: display window contents.
    .
    .
    .
  return(0).
```

**Application
Delete Call**

8.5.3 As with the Display call, the screen, window, and user-defined numbers are passed to the application Delete call. Through this call, anything displayed by the application Display routine is deleted from the video device. See the example Delete Call routine that follows.

```
APPDEL(SCREEN#,WINDOW#,UDWNBR)
  get the window coordinates.
  case UDWNBR of
    10: delete window contents.
    20: delete window contents.
    .
    .
    .
  return(0).
```

**Application
Message
Manager Call**

8.5.4 The application Message routine requires several parameters to be able to handle the different situations where it may be called. Regardless of when it is called, the message screen number and message window number will be passed. These two numbers provide a means of identifying the actual window used to display the message. Also passed for every message are the message number and the message type assigned in Message Builder.

If APPMSG is called to handle a Help message, the screen, window, and item numbers are passed in to identify where the cursor was when the Help key was pressed. The item parameter will have a value of -1 if window-level help is being displayed. These three parameters have no meaning for Error, Warning, and Please Note messages.

When returning to Window Manager, the application Message routine must pass back the key code for the key pressed by the user.

```
APPMSG(MSGSCR#,MSGWND#,KEY,MSGNBR,MSGTYP,SCREEN#,WINDOW#,ITEM#)
  get the help window coordinates.
  case msgnbr of
    25: display message.
    26: display message.
    .
    .
  get user input:
  if select key,
    key param = select key.
    delete message window.
    return(0).
  if (backup key or help key) and (msg type = help),
    key param = key pressed.
    delete message window.
    return(0).
  go to get user input.
  return(0).
```

**Making
Window
Manager
Calls Inside
a UDW Routine**

8.6 All the standard Window Manager calls can be made inside any of the UDW routines. Other windows can be displayed, received from, and changed. It is recommended that you not make recursive calls on the UDW (such as in APPRCV calls, by making a WMWRCV call on the same window). This can result in wasted time, as the application routine will be called again.

INTERNAL PHRASE EDITING

9

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
9.1	Introduction	9-3
9.2	Examples of Internal Phrases.....	9-3
9.3	Modifying Internal Phrases	9-4
9.3.1	Modifying Phrases Before Linking the Application	9-4
9.3.2	Modifying Phrases Using a Phrase Input File (Phrase Editor Utility).....	9-4
9.4	Using PBUILD.....	9-5
9.4.1	Phrase Editor Commands	9-5
9.4.1.1	Edit a Phrase	9-6
9.4.1.2	Restore Default Phrases.....	9-6
9.4.1.3	Print Phrases to File	9-6
9.4.1.4	Save Phrases to File	9-7
9.4.1.5	Exit	9-7
9.5	Usage of the NLXPHRAS.NM\$ File	9-7
9.6	Listing of Internal Phrases.....	9-7

Introduction

9.1 Window Manager (and Toolkit, if you are using it) display(s) several windows not directly controlled by the application developer. These windows include those used to display Help and error messages. The developer can change the phrasing in these windows to suit the language of the user or the developer's tastes. Internal Phrase Editing allows this capability. This chapter discusses the following:

- Phrases that can be modified
- Steps for modifying phrases before linking the application
- Steps for modifying phrases by using an input file at run time.

Examples of Internal Phrases

9.2 Internal phrases are those that the application developer cannot create or edit through Screen Builder or through calls to Window Manager. They consist of internal Error messages not usually accessed by the developer, such as Help, Error, and Press-the-ENTER-key-to-continue labels in Help and Error windows. See Figures 9-1 and 9-2 for examples.

Figure 9-1

Internal Error Message

```
*** ERROR ***

The message file

        PBERRMSG.NM$

could not be found in the current working
directory. To see the text for message
number

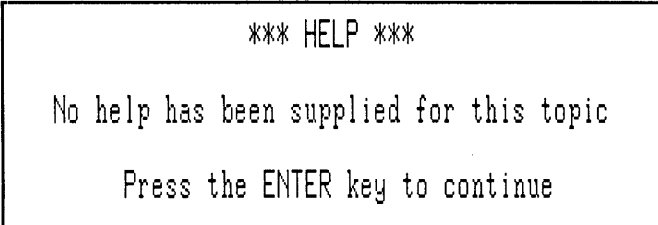
        6

load a copy of the message file onto the
current working directory.

Press the ENTER key to continue
```

Figure 9-2

Help Message



```
*** HELP ***  
  
No help has been supplied for this topic  
  
Press the ENTER key to continue
```

**Modifying
Internal
Phrases**

9.3 There are two methods available for modifying internal phrases, one for use when linking and the other at run time. The linking method, in which one of the Window Manager source files is edited and reassembled, is for use by developers who are limited in disk space, want to change the phrase permanently, save load time or data space, or wish to control the number of support files needed by their application. The run time method, in which a special file is created, is for developers whose applications need to be modified for different users but who do not wish to relink a special version for each. The run-time method is good to use for modifying the application in foreign languages, for example.

**Modifying Phrases
Before Linking the
Application**

9.3.1 This option requires reassembly of the source file WMSTRDEF.ASM, which is found on the Window Manager Run-time Object disk. WMSTRDEF.ASM contains the text of all phrases listed in Tables 9-1 and 9-2 (located at the end of this chapter). You can change the text of the phrases within quotes, using a text editor. Comments in the WMSTRDEF.ASM file will tell you where each phrase is used, and the maximum number of characters each phrase can contain. After assembling the source, WMSTRDEF.OBJ should then be linked with the application. (See the Appendix for the language you are using for information on linking files.) This edited version will then be the reference for all internal phrases.

**Modifying Phrases
Using a
Phrase Input File
(Phrase Editor
Utility)**

9.3.2 The Phrase Editor is an interactive utility that aids in constructing a file of internal phrases. This internal phrase file is read at run time, and the phrases contained in this file override the default phrases specified in WMSTRDEF.OBJ.

The Phrase Editor utility can be used to modify any of the internal phrases. You can modify only the subset of phrases used by Window Manager or the entire range of Window Manager and Toolkit phrases.

Using PBUILD

9.4 The Phrase Editor is invoked by the PBUILD command. Before invoking PBUILD, you should use the MS-DOS SET command to set the environment variable NLXTOOLS. NLXTOOLS should be set to the directory path where the screen files and message files for the PBUILD utility have been placed (.PIC, .NS\$, and .NM\$ files). If NLXTOOLS is not set, all of the PBUILD support files must be placed in the default directory. Setting NLXTOOLS to equal a MS-DOS directory path enables you to place the support files in a separate directory.

When PBUILD is invoked, it looks in the default directory for the internal phrase file NLXPHRAS.NM\$. If it is present, PBUILD reads it and displays the current set of phrases. If it is not present, a message is displayed to that effect and the default English phrases are displayed for editing. The name of the internal phrase file, NLXPHRAS.NM\$, must not be changed since this is the name searched for at run time. NLXPHRAS.NM\$ must never be altered except by PBUILD.

Phrase Editor Commands

9.4.1 Once PBUILD has been invoked, a window appears asking you to make a choice between Window Manager phrases and Toolkit/Window Manager phrases. Your choice depends on your application. If you are not using Toolkit, choose the smaller subset (that is, Window Manager phrases). After you have made a choice, the screen shown in Figure 9-3 appears, displaying the various commands available for PBUILD.

Figure 9-3

Phrase Editor Commands Screen

NATURALLINK PHRASE EDITING UTILITY	TOOLKIT/WINDOW MANAGER PHRASES
Edit a Phrase	Save Phrases to File
Restore Default Phrases	Exit
Print Phrases to File	
<u>PHRASES</u>	
*** HELP ***	
*** ERROR ***	
*** WARNING ***	
*** PLEASE NOTE ***	
The message file could not be found in the specified working directory. To see the text for message number load a copy of the message file onto the specified working directory.	
No help has been supplied for this topic. ENTER: Stop Help HELP KEY: More Help Press the ENTER key to continue	
An error with the following code was encountered:	
Press ESC for command menu	Press F7 for Help

Edit a Phrase 9.4.1.1 When the Edit a Phrase command is chosen, the cursor is placed in the PHRASES window. The PHRASES window can be scrolled to list all of the phrases. When a phrase is selected, a pop-up window, Figure 9-4, appears so you can edit the phrase. The Help messages attached to the phrases will explain where the phrase is used in the NaturalLink run-time.

Pressing the **ENTER** key ends the editing of the phrase and returns the cursor to the PHRASES window. Pressing the **ESC** key while in the PHRASES window returns you to the Command window. Pressing **F7** invokes the display of online Help messages.

Figure 9-4

Enter Phrase Pop-Up Window

NATURALLINK PHRASE EDITING UTILITY	TOOLKIT/WINDOW MANAGER PHRASES
Edit a Phrase Restore Default Phrases Print Phrases to File	Save Phrases to File Exit
*** HELP ***	
<u>PHRASES</u>	
<u>Enter Phrase</u>	
*** PLEASE NOTE *** ----- Press ESC to Abort	
message number load a copy of the message file onto the specified working directory. No help has been supplied for this topic. ENTER: Stop Help HELP KEY: More Help Press the ENTER key to continue An error with the following code was encountered:	
Press ESC for command menu	Press F7 for Help

Restore Default Phrases 9.4.1.2 When the Restore Default Phrases command is chosen, all of the original default phrases are displayed. Regardless of the presence of the NLXPHRAS.NM\$ file, edited phrases are not displayed, and changes made and not saved are lost. The cursor remains in the Commands window.

Print Phrases to File 9.4.1.3 This option can be used to get a text listing of all your internal phrases. When the Print Phrases to File command is chosen, a pop-up window appears asking for the pathname to which the phrases will be printed. This is different from the NLXPHRAS.NM\$ file in that the phrases are output in text format.

Save Phrases to File 9.4.1.4 When the Save Phrases to File command is selected, PBUILD creates the NLXPHRAS.NM\$ file in the default directory. A pop-up window appears to inform you that this was done correctly. If a NLXPHRAS.NM\$ file is already present, it will be written over rather than appended.

The saved NLXPHRAS.NM\$ file contains only the phrases that have been modified. This saves file size and time when the file is loaded at run time.

Exit 9.4.1.5 When the Exit command is selected, PBUILD checks to see if any phrases have been changed. If they have, a pop-up window appears asking if you wish to save these changes. Selecting yes causes the Save command to be executed before PBUILD terminates. Selecting no causes PBUILD to terminate without saving any changes. If no phrases have been changed, PBUILD quits immediately.

Usage of the NLXPHRAS.NM\$ File

9.5 At application run time, Window Manager will attempt to load the NLXPHRAS.NM\$ file when the INITIALIZE call is made. If the file is not found, the phrases in the WMSTRDEF.OBJ module are used (the linked-in phrases). If the NLXPHRAS.NM\$ file does exist, the phrases it contains are substituted for the appropriate phrases in WMSTRDEF.

Window Manager, by default, will look for the NLXPHRAS.NM\$ file on the default drive and directory. However, the application may place the NLXPHRAS.NM\$ file on another drive or directory and then tell Window Manager where it is by setting the Llwmpath variable (wmpath for FORTRAN users). Llwmpath is a string variable which may be set using the Set String Value call. The Set String Value call must be made before the Initialize call. Dummy screen, window, and item parameters may be passed, as Window Manager will not check them when you are setting Llwmpath. If Llwmpath is set, Window Manager will append the directory path specified by Llwmpath to the NLXPHRAS.WM\$ filename to get the full pathname of the phrase file. Llwmpath should contain only the value of a directory path; a filename should not be included. Also remember that if Llwmpath is set, Window Manager also looks for the help files associated with the application's screens in the directory specified by Llwmpath.

Listing of Internal Phrases

9.6 Table 9-1 provides a list of Window Manager internal phrases that can be modified, and their location. Table 9-2 provides a similar list for Toolkit. These lists are in the same order as the phrases are displayed to the developer when they are modified.

Table 9-1 Window Manager Internal Phrases That Can Be Modified

Headers for Message Windows

*** HELP ***
*** ERROR ***
*** WARNING ***
*** PLEASE NOTE ***

Default Message File Error Messages

The message file
could not be found in the specified
current working directory. To see the text for
message number
load a copy of the message file onto the
current working directory.

Default Help Message

No help has been supplied for this topic

Footers for Message Windows

ENTER: Stop Help HELP KEY: More Help
Press the ENTER key to continue

Default Message Processing Error Messages

An error with the following code was encountered:
The message display system was unable to
neatly process this message for the following reason:
message files read error
invalid error file name passed to Message Manager
The following information is known about the error:

An invalid message number has been supplied
to message manager.
The invalid number is:

Table 9-2 Toolkit Internal Phrases That Can Be Modified

Labels and Text for Expert Windows

JAN
FEB
MAR
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
Enter
:
Day:
Month:
Year:
Enter @1 in the @2 range @3-@4@5
in an increment of @1:
:
inclusive
exclusive
new value

Text for Command Window Help Item

Help is available for any item on the screen.
To get help, move the cursor to the item and
press the Help key.

Text for Show Translation Window

Command translation:

Parser Memory Error Message

This sentence is too large for the available
memory space. Please use the Rubout/Backup
key to shorten your sentence to a point where
it is executable, or try to split the command
you want into two shorter sentences.

Busy Message

Working. Please wait...
...

Table 9-2 Toolkit Internal Phrases That Can Be Modified (Continued)

Text for Saved Sentences Process

Select a sentence to recall or press the Quit key to abort
Select a sentence to delete or press the Quit key to abort
A name must be entered.
This name has already been used.
Enter a name for the saved sentence:
Press the QUIT key to abort
Do you wish to
Save
Recall
Delete
a sentence?

Date Expert Error Text

Date entered not within range.
Day entered not within range.
Month entered not within range.
Year entered not within range.

Text for Interface Customization Process

PRESS: F7 for Help F10 to Proceed ESC to Quit
Editing the phrase:
Adding a synonym to the phrase:
PRESS: F7 for Help ESC to Quit
I
want to
add synonyms to visible windows
add synonyms to pop-up windows
edit phrases in visible windows
edit phrases in pop-up windows
remove selected synonyms or edited phrases
remove all synonyms and edited phrases
look at the screen
quit
Remove all synonyms and edited phrases?
yes
no
Adding Synonyms or Editing Phrases

Table 9-2 Toolkit Internal Phrases That Can Be Modified (Continued)

Press ENTER to select the phrase that you wish to edit or create a synonym of.

Use the CTRL-ARROW and CTRL-HOME keys to move between windows.

Press F10 when you're through and want to go back to the menu window or, if editing pop-up windows, you want to proceed to the next window.

Press ESC when you want to quit what you're doing.

Press ENTER now to get out of this window.

Removing Synonyms or Edited Phrases

The original phrase for

The phrase

is

is a synonym of

Press ENTER to select a synonym or edited phrase to be removed.

Use the CTRL-ARROW and CTRL-HOME keys to move between windows.

Press F10 when you're through and want to proceed to the next pop-up window or, if there are no more pop-ups, return to the menu.

Press ESC when you want to quit what you're doing.

Press ENTER now to get out of this window.

Do you want to save your changes?

Saving Changes.

Please Wait.

10

WINDOW MANAGER INPUT DEVICES

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
10.1	Overview	10-3
10.2	Window Manager Keyboard Input	10-3
10.2.1	Window Manager Keyboard Input Values	10-4
10.3	Window Manager Input Default Functions.....	10-4
10.3.1	Select — Default: ENTER Key	10-7
10.3.2	Proceed — Default: F10 Key	10-7
10.3.3	Page Scroll — Default: F1 and F2 Keys.....	10-7
10.3.4	Help — Default: F7 Key.....	10-8
10.3.5	Backup — Default: F8 Key	10-8
10.3.6	Next Item/Line Scrolling — Default: Arrow Keys	10-8
10.3.7	Move Right/Left One Item — Default: TAB/SHIFT-TAB Key.....	10-9
10.3.8	Top/Bottom — Default: HOME Key	10-9
10.3.9	Next Window — Default: CTRL-L/R Arrow Keys and CTRL-PgUp/PgDn Keys.....	10-9
10.3.10	Next Active Window — Default: CTRL-HOME Key.....	10-9
10.3.11	Move to Start/End of Line — Default: CTRL-F1/F2 Keys	10-9
10.3.12	Move Left or Right One Word — Default: CTRL-F3/F4 Keys	10-9
10.3.13	Insert — Default: INS Key.....	10-10
10.3.14	Delete — Default: DEL Key	10-10
10.3.15	Delete From the Cursor Out — Default: CTRL-F5 Key.....	10-10
10.3.16	Backspace — Default: BACKSPACE Key.....	10-10
10.3.17	Printable Keys — Default: Keys With Key Codes Between 00H and FFH	10-10
10.3.18	Using Printable Keys in the Search Mode.....	10-11
10.3.19	Input Unknown to Window Manager	10-11
10.4	Changing Function Key Assignments.....	10-11
10.5	Application Input Routine	10-13
10.5.1	Definition and Use.....	10-14
10.5.2	APPINP Return Code	10-14
10.5.3	Dummy APPINP Routine	10-14

Overview

10.1 Window Manager handles function key, cursor-movement key, and control key input. This chapter explains the following items:

- Keyboard input
- Window Manager operations performed by keyboard input
- Function Key Change utility
- Application input routine

Tables are provided showing Window Manager operations for both inactive and active windows. Tables showing key codes for specific machines can be found in Appendix C, Computer-Specific Information.

Window Manager Keyboard Input

10.2 Window Manager uses only a subset of the keys available on any computer. The code values of the keys in this subset are defined to Window Manager, and input from this subset is used to perform specific Window Manager operations such as Select, Scroll, Proceed, and so on. These keys are referred to as Window Manager function keys. Pressing keys *not* defined to Window Manager (that is, keys whose key code values are not recognized by Window Manager) causes a return to the application program. These keys are referred to as unknown keys in the context of Window Manager functionality.

Any key that the user presses is the fourth parameter of the Window Manager Receive call. The value of any key that causes the Window Manager Receive call to terminate is referred to as the key return value. In the majority of cases, this is the value of the Select key (the default is the **ENTER** key), but any unknown key value could be returned. It is *very* important for the application program to check this key parameter after every Receive call.

**Window Manager
Keyboard
Input Values**

10.2.1 In order to interpret the key value correctly, it is necessary to understand how Window Manager handles keyboard input. When Window Manager performs keyboard input/output (I/O), it makes every key value unique and returns these unique key values to the application program. Two rules apply:

1. If the key is a normal ASCII character (that is, if the key code is returned in register AL with the key number in AH), Window Manager returns the exact key value. For example, the key code for the **ENTER** key is 0DH. Thus, 0DH is returned by Window Manager.
2. If the key is a special function (non-ASCII character) which returns an extended code (that is, register AL = 0, with extended code in AH), Window Manager returns the extended code plus 100H. For example, the key code for the **F10** key is the extended code 44H. Therefore, Window Manager returns 144H if this key is pressed.

**Window
Manager
Input Default
Functions**

10.3 Tables 10-1 and 10-2 provide a quick reference list of the default key assignments for both inactive and active windows. Display windows and user-defined windows (UDWs) are not included in the tables. Display windows are not listed because the cursor is not allowed in a display window and information cannot be received from one. UDWs are not listed because the application controls how keys in these windows are used. The paragraphs following Tables 10-1 and 10-2 explain each of the listed functions. The default keys vary depending on the computer you are using. Consult Appendix C for a list of key differences.

Table 10-1 Default Keys in Inactive Windows

<i>Function Keys</i>	<i>List Window</i>	<i>Edit Window</i>	<i>Text and File Windows</i>
ENTER (select)	Invalid, beep	Invalid, beep	Invalid, beep
F10 (proceed)	Return to calling routine	Return to calling routine	Return to calling routine
F1 and F2 (page scroll)	Scroll window	Scroll window	Scroll window
F7 (help)	Help for item cursor is on	Help for item cursor is on	Help for window cursor is in
F8 (backup)	Show previous help message or return	Show previous help message or return	Show previous help message or return
Arrow Keys (move cursor)	Move cursor to next item	Move cursor to next item	Scroll window a line at a time

Table 10-1 Default Keys in Inactive Windows (Continued)

<i>Function Keys</i>	<i>List Window</i>	<i>Edit Window</i>	<i>Text and File Windows</i>
HOME (top/bottom)	Move cursor to top/bottom item	Move cursor to top/bottom item	Move text to show top/bottom
CTRL L/R ARROW CTRL PgUp/PgDn (move cursor)	Move cursor to adjacent window	Move cursor to adjacent window	Move cursor to adjacent window
CTRL HOME (move cursor)	Move cursor to next active window	Move cursor to next active window	Move cursor to next active window
CTRL F1/F2 (move cursor)	Invalid, beep	Invalid, beep	Invalid, beep
CTRL F3/F4 (move cursor)	Invalid, beep	Invalid, beep	Invalid, beep
TAB/SHIFT TAB (move cursor)	Move cursor right/left one item	Move cursor right/left one item	Invalid, beep
INS (insertion)	Invalid, beep	Invalid, beep	Invalid, beep
DEL (deletion)	Delete last key searched for	Invalid, beep	Invalid, beep
CTRL F5 (deletion)	Invalid, beep	Invalid, beep	Invalid, beep
BACKSPACE (move cursor)	Invalid, beep	Invalid, beep	Invalid, beep
Printable keys Keys > 0x00 Keys ≤ 0xFF	Enter search mode	Invalid, beep	Invalid, beep
Any Other Key (return keys)	Return to calling routine	Return to calling routine	Return to calling routine

Table 10-2 Default Keys in Active Windows

Function Keys	List Window	Edit Window	Text and File Windows
ENTER (select)	Return to calling routine	Return to calling routine	Return to calling routine
F10 (proceed)	Return to calling routine	Return to calling routine	Return to calling routine
F1 and F2 (page scroll)	Scroll window	Scroll window	Scroll window
F7 (help)	Help for item cursor is on	Help for item cursor is on	Help for window cursor is in
F8 (backup)	Show previous help message or return	Show previous help message or return	Show previous help message or return
ARROW KEYS (move cursor)	Move cursor to next item	Move cursor to next item	Scroll window a line at a time
HOME (top/bottom)	Move cursor to top/bottom item	Move cursor to top/bottom item	Move test to show top/bottom
CTRL L/R ARROW CTRL PgUp/PgDn (move cursor)	Move cursor to adjacent window	Move cursor to adjacent window	Move cursor to adjacent window
CTRL HOME (move cursor)	Move cursor to next active window	Move cursor to next active window	Move cursor to next active window
CTRL F1/F2 (move cursor)	Invalid, beep	Move cursor to the beginning or end of line	Invalid, beep
CTRL F3/F4 (move cursor)	Invalid, beep	Move cursor one word left or right in line	Invalid, beep
INS (insertion)	Invalid, beep	Allow letters to be inserted	Invalid, beep
TAB/SHIFT TAB (move cursor)	Move cursor right/left one item	Move cursor right/left one item	Invalid, beep
DEL (deletion)	Delete last key search for	Delete letter cursor is on	Invalid, beep
CTRL F5 (deletion)	Invalid, beep	Delete line from cursor out	Invalid, beep

Table 10-2 Default Keys in Active Windows (Continued)

<i>Function Keys</i>	<i>List Window</i>	<i>Edit Window</i>	<i>Text and File Windows</i>
BACKSPACE (move cursor)	Invalid, beep	Move cursor backwards one character	Invalid, beep
Printable keys Keys > 0x00 Keys ≤ 0xFF	Enter search mode	Character entered	Invalid, beep
Any Other Key (return keys)	Return to calling routine	Return to calling routine	Return to calling routine

**Select —
Default: ENTER Key**

10.3.1 Select is probably the most important of all Window Manager functions. It is the primary means of specifying commands to the application program. This function is used to choose items from a window; it is valid only in active windows. When Select is invoked either in text and file windows or in single-selection list and edit windows, Window Manager returns control to the application program. If the window is a multiple-selection edit window, the cursor moves from item to item until it reaches the last item in the window, at which time it returns to the application program. If Select is invoked in a multiple-selection list window, the Chosen/Enable attribute of the item the cursor is on is set to yes (1) and the cursor moves down to the next selectable item in the window.

**Proceed —
Default: F10 Key**

10.3.2 The Proceed key causes Window Manager to return to the application program from the window in which the cursor is located. Window Manager uses this function to commit all selections in a multiple-selection window, but an application can also use the Proceed key as a Quit key (if it has been so defined), since Window Manager always returns to the application program when the Proceed key is pressed.

**Page Scroll —
Default:
F1 and F2 Keys**

10.3.3 These keys cause Window Manager to scroll the window containing the cursor by the number of lines in that window. For example, if a window can display 10 lines of information, that information moves up or down 10 lines when the page-scrolling function is invoked. The page-scroll-up operation (default — F1 key) causes the text in the window to move up. The page-scroll-down operation (default — F2 key) causes the text in the window to move down.

**Help —
Default: F7 Key**

10.3.4 When the Help function is invoked, Window Manager first tries to display the Help message attached to the item the cursor is on. If no Help message exists for this item, Window Manager tries to display the help message attached to the window. If more than one Help message is attached to either an item or a window, pressing the Help key while a Help message is shown causes the next attached message to appear. Pressing the Backup key causes the previous Help message to be displayed.

If the first Help message is displayed, pressing the Backup function results in return to the screen that was displayed before help was requested. The Select function accomplishes the same results.

**Backup —
Default: F8 Key**

10.3.5 The Backup function causes Window Manager to back up one step in the current procedure. If this function is used in conjunction with a series of Help messages attached to a single item or window, it causes the previous Help message to be redisplayed.

NOTE: The Backup key has meaning to Window Manager only when a Help message is displayed. If the Backup key is pressed at any other time, Window Manager returns to the application program.

**Next Item/Line
Scrolling —
Default: Arrow Keys**

10.3.6 The arrow keys cause the cursor to either move item by item in a window or, in text and file windows, to scroll the window line by line. The cursor moves in the direction the **arrow** key points.

In edit windows, the **left** or **right arrow** keys move the cursor a character at a time in an edit field. If the arrow keys move the cursor over an empty edit field, the censored spaces become fill characters. Before returning to the application program, Window Manager converts the fill characters to blanks when they are followed by text. However, trailing fill characters (those following text) are removed.

**Move Right/Left
One Item — Default:
TAB/SHIFT-TAB
Key**

10.3.7 The **TAB** key moves the cursor one item to the right; the **SHIFT-TAB** key moves the cursor one item to the left. These keys are valid for movement only in a multiple column or free-format window. They perform the same function for an edit window as the **left** and **right arrow** keys perform for list windows. In edit windows these keys move the cursor one item left or right, whereas the **left** or **right arrow** keys move the cursor one character left or right.

**Top/Bottom —
Default: HOME Key**

10.3.8 This function key moves the cursor to the first item in the window; if it is already on the first item, it moves the cursor to the last item in the window. For text and file windows, Window Manager moves the text so the first or last line in the window is showing.

**Next Window —
Default: CTRL-L/R
Arrow Keys and
CTRL-PgUp/PgDn
Keys**

10.3.9 The Next Window function keys move the cursor from one window to another. The cursor is moved to the closest window in the direction of the key pressed.

**Next Active
Window — Default:
CTRL-HOME Key**

10.3.10 Several windows can be visible to the user at once, and more than one of those windows can be active. The Next Active Window function provides a rapid means of moving from one active window to another. When invoked, the cursor moves from active window to active window based on the order in which the windows were added.

**Move to Start/End
of Line — Default:
CTRL-F1/F2 Keys**

10.3.11 In edit windows it is useful to move quickly to the beginning or end of the edit field. This can be performed in Window Manager with the use of the **CTRL-F1** key, which moves the cursor to the beginning of the current edit field, and the **CTRL-F2** key, which moves the cursor to the end of the text in the current edit field. These functions are valid only in edit windows.

**Move Left or Right
One Word — Default:
CTRL-F3/F4 Keys**

10.3.12 This function causes the cursor to move through an edit field one word at a time. The **CTRL-F3** key moves the cursor one word to the left in the current edit field; the **CTRL-F4** key moves the cursor one word to the right in the current edit field. These functions are valid only in edit windows.

**Insert —
Default: INS Key**

10.3.13 The Insert key is valid only in an active edit window. When the Insert key is pressed, Window Manager allows characters to be inserted in front of the cursor. Window Manager remains in the insert mode until a nonprintable character is pressed, such as an **arrow** key or the DEL key.

**Delete —
Default: DEL Key**

10.3.14 When used in an edit window, this function causes Window Manager to delete the character on which the cursor is positioned. The Delete key can also be used as part of the Window Manager search feature. In that case, the Delete key deletes the last key code entered in a search string. For example, if the user types in the characters ABC and then presses the Delete key, Window Manager searches for an item beginning with only the characters AB.

**Delete From
the Cursor Out —
Default: CTRL-F5
Key**

10.3.15 Sometimes it is useful to delete several or all of the characters in an edit field quickly. The **CTRL-F5** key deletes all characters right of the cursor to the end of the edit field, including the character the cursor is on. If **CTRL-F5** is pressed when the cursor is at the beginning of the edit field, all text in that field is deleted but the edit field itself still exists and the cursor remains in place.

**Backspace —
Default:
Backspace Key**

10.3.16 The Backspace key is valid only in an active edit window. When the Backspace key is pressed, the cursor moves backward one character.

**Printable
Keys — Default:
Keys With Key
Codes Between
00H and FFH**

10.3.17 These keys are valid only in list and edit windows. In an edit window, a printable key is entered as part of the current edit field. In a list window, pressing a printable key causes Window Manager to enter the search mode.

Printable keys also can be used to cause Window Manager to return control to the application program. Ten such printable keys can be defined using the Function Key Change utility.

**Using Printable Keys
in the Search Mode**

10.3.18 In the search mode, Window Manager looks for an item that starts with the character(s) pressed. It does this by building a search string from the character keys pressed, putting the cursor on the first item that starts with the characters in the search string. If no match is found, a beep sounds. Window Manager continues to build upon the same search string until an unprintable key (such as an **arrow** key) is pressed.

For example, if the user types A, Window Manager finds the first item starting with A and puts the cursor on it. If B is then pressed, it continues the search by looking for an item starting with AB. Now if the character key C is pressed and an item that starts with ABC cannot be found, a beep sounds and the character C is not added to the search string. Once an unprintable character key is pressed, the search string is cleared and a new search can start.

**Input Unknown
to Window Manager**

10.3.19 If input from the keyboard returns a value that is unknown to Window Manager, Window Manager returns to the application program.

**Changing
Function Key
Assignments**

10.4 Depending on application program requirements, you may wish to change the default function keys used by Window Manager. To change the assigned function key values, execute the function key change utility KBUILD. As with the SBUILD and MBUILD utilities, the MS-DOS SET command can be used to set the NLXTOOLS environment variable to the directory path containing KBUILD support files.

This utility presents the list of assigned functions and allows you to specify the keys to be assigned to various functions. When you make your assignments and commit them, the utility generates the object file WMKEYDEF.OBJ on the current drive and directory. You must then explicitly include WMKEYDEF.OBJ in your link stream. The KBUILD Utility Menu is shown in Figure 10-1. Refer to Appendix C, Computer-Specific Information, for a listing of the computers referenced in the TIPC and OTHER columns in Figure 10-1.

Figure 10-1

KBUILD Utility Menu

SET FUNCTION KEYS UTILITY		
Which product are you setting the function keys for?		
<u>WINDOW MANAGER ONLY</u>	NATURALLINK TOOLKIT AND WM	
	TIPC	Other
Select items :	0D	0D
Proceed to next step :	144	144
Move cursor up one item :	148	148
Move cursor down one item :	150	150
Move cursor left one item :	14B	14B
Move cursor right one item :	14D	14D
Move to the top/bottom item :	147	147
Move cursor up one window :	184	184
Move cursor down one window :	176	176
Move cursor left one window :	173	173
Move cursor right one window :	174	174
Move to the next active window :	177	177
Up page scroll :	13B	13B
Down page scroll :	13C	13C
Press F7 for help, F10 to commit the entries, or ESC to abort.		

This is the only menu for the function key utility. Since the NaturalLink Window Manager is related to other NaturalLink products, you must specify the NaturalLink product for which you are setting the function keys, Window Manager only, or both Window Manager and Toolkit. Selecting the NATURALLINK TOOLKIT AND WM option causes additional functions specifically used by the NaturalLink Toolkit to be presented in the key-codes window. Key codes can be assigned to these functions.

Once the option has been selected, the key-codes window displays the functions that can be performed with the function keys. The values listed in the TIPC and OTHER columns in Figure 10-1 will come from an existing WMKEYDEF.OBJ file or will be initialized by the KBUILD utility itself. The utility first looks for a WMKEYDEF.OBJ file on the default drive and directory. If the file exists, the previously specified values in WMKEYDEF.OBJ are presented. If the KBUILD utility has never been used or if the previous values cannot be found, an error message is displayed indicating that the file cannot be found or that there is a problem with the file. The default values shown in Figure 10-1 (also listed in the Help messages) are then used.

Values can be entered for both TI and other computers. These values must be the values of the keys that invoke the function you intend. Enter the hexadecimal input values according to these guidelines:

- If the key you wish to assign to the function is a normal ASCII character (that is, a key code returned in register AL with the key number in AH), enter the exact key value. For example, the key code for the **ENTER** key is 0DH; thus 0D is entered as the value.
- If the key you wish to assign is a special function key (non-ASCII code) which returns an extended code (that is, register AL = 0 with the extended code in AH), enter the extended code plus 100H. For example, the key code for the **F10** key is the extended code 44H; therefore, enter 144 for the value.
- If you do not want Window Manager to perform a particular function, enter 0 (zero) as the value for that function.

Once you have entered all the key codes, press the **F10** key. The function key utility then generates the object file WMKEYDEF.OBJ on the default drive and directory.

IMPORTANT: If WMKEYDEF.OBJ already exists on the default directory, it *will be replaced* by the new file being generated. After the file has been created, you are returned to the operating system level.

To put your function key changes into your application, link the version of WMKEYDEF.OBJ that was generated by KBUILD rather than the default version included in the Window Manager run-time library (WM.LIB) on the NaturalLink object diskette. Once linked, the new function key assignments are used by Window Manager and the NaturalLink software until you change them and relink.

Application Input Routine

10.5 The Window Manager primary means of getting input from the user is through the keyboard. The developer, however, may want to provide an alternative method of user input, such as a mouse, lightpen, or speech. Window Manager does not directly support these input devices. Instead, a generic application input routine can be used.

Definition and Use 10.5.1 The application input routine takes the following form:

```
KEY = APPINP()
```

This routine must be provided by the developer. APPINP is called during the Receive call. If no key is returned (a value of zero is returned) from APPINP, the keyboard is polled. This polling goes back and forth between input methods until one of them has a key to input. The pseudocode for Window Manager input is as follows:

```
GETINPUT()
  key = 0
  while (key == 0)
    key = appinp()
    if (key == 0)
      if keyboard has key ready
        get key from keyboard
  return(key)
```

APPINP Return Code 10.5.2 Any input method can be used by the application for input to Window Manager. However, the application must first translate the input into the form Window Manager expects. Window Manager always expects APPINP to return a value in the form of a key code. This means that the APPINP routine will act as another version of the keyboard, a transparent keyboard if you will.

Also, the key codes returned must be the same as the codes assigned to the Window Manager functions via the KBUILD utility. If they are not, Window Manager assumes the key is unknown and returns to the application program. For example, if the application receives a Select function from the input device it is controlling, the key code for the Select function from the keyboard must be returned by the application input routine.

If APPINP has no input ready when Window Manager calls it, a value of zero (0) should be returned to indicate that no key is ready. Window Manager then polls the keyboard for input. If a 0 is not returned from APPINP, Window Manager will wait indefinitely for input from the application input routine.

Dummy APPINP Routine 10.5.3 A dummy application input routine has been provided with the Window Manager run-time object code to resolve the reference to the APPINP call. This dummy routine can be linked in with the application if no special input routine will ever be needed by the application. The dummy routine is shipped as part of a library of dummy routines. Consult the appropriate appendix for the language you are using for details.

The dummy routine always returns a value of 0 (zero) indicating there is no input ready.

EQUIPMENT REQUIREMENTS

A

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
A.1	Equipment Requirements.....	A-3
A.1.1	Equipment Necessary for Interface Development	A-3
A.1.2	Equipment Necessary for User Operation	A-4

Equipment Requirements

A.1 The equipment requirements for using Window Manager differ depending upon whether an interface application is being developed or is merely being run.

Equipment Necessary for Interface Development

A.1.1 To use Window Manager and associated utilities for interface development, the application designer needs the following equipment.

- Texas Instruments Professional Computer (TIPC), Texas Instruments Portable Professional Computer (TIPPC), Texas Instruments BUSINESS-PRO™ Professional Computer, IBM® PC, IBM PC/XT™, or IBM Personal Computer AT™, with at least two disk drives and a minimum of 256K RAM, where K equals 1024 bytes
- MS-DOS (up to and including Version 3.xx)
- The MS-DOS link editor or a compatible link editor
- One of the following programming languages:
 - Lattice C Compiler (up to and including Version 2.14)
 - MS-FORTRAN (up to and including Microsoft Version 3.2 or a compatible version for the machine you are using)
 - MS-Pascal (up to and including Microsoft Version 3.2 or a compatible version for the machine you are using)
 - MS-BASIC Compiler (TI Version 1.0 or a compatible version for the machine you are using)

BUSINESS-PRO is a trademark of Texas Instruments Incorporated.

IBM is a registered trademark of International Business Machines Corporation.

IBM PC/XT and IBM Personal Computer AT are trademarks of International Business Machines Corporation.

**Equipment
Necessary for
User Operation**

A.1.2 If the application requires Window Manager only, the user needs the following equipment:

- TIPC, TIPPC, TI BUSINESS-PRO, Texas Instruments PRO-LITE™ Professional Computer, IBM PC, IBM PC/XT, or IBM Personal Computer AT.
- MS-DOS (up to and including Version 3.xx).
- A minimum of 80K bytes of RAM (this allows for a Window Manager data space of approximately 32K bytes; your application may need more or less space than this), plus any additional memory required by the application itself. Whether the user's machine needs a Winchester drive depends on the application software, not the presence of Natural-Link software.

PRO-LITE is a trademark of Texas Instruments Incorporated.

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
B.1	Introduction	B-3
B.2	Window Types	B-3
B.2.1	List Windows.....	B-3
B.2.2	Text Windows.....	B-4
B.2.3	Display Windows	B-4
B.2.4	Edit Windows.....	B-4
B.2.4.1	Maximum Edit Field Length.....	B-4
B.2.4.2	Required Edit Field.....	B-5
B.2.4.3	Echo Edit Field Input	B-5
B.2.4.4	Multiple Selection in an Edit Window	B-5
B.2.5	File Windows	B-5
B.3	Window Labels.....	B-6
B.3.1	Top Window Label	B-6
B.3.2	Bottom Window Label.....	B-7
B.3.3	Left Window Label.....	B-7
B.3.4	Right Window Label	B-7
B.4	Window Formats.....	B-8
B.4.1	Single-Column Format.....	B-9
B.4.2	Multiple-Column Format	B-9
B.4.3	Free Format	B-10
B.5	Item Labels.....	B-11
B.5.1	Maximum Item Label Length	B-12
B.5.2	Left-Justified Labels	B-12
B.5.3	Right-Justified Labels.....	B-12
B.5.4	Free-Format Labels	B-12
B.6	Special Window Manager Features.....	B-13
B.6.1	Use of Borderless Windows and Multiple Window Effects	B-13
B.6.1.1	Adding Multiple Windows in Order	B-14
B.6.1.2	Reasons for Using Multiple Windows.....	B-14
B.6.2	Use of Display Windows	B-15
B.6.3	Creating and Using Blank Lines	B-16
B.6.4	Unselectable Items.....	B-16
B.6.5	Use of Invisible Items.....	B-17
B.6.6	When to Use Multiple-Selection Windows	B-17

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
B.6.7	Simulating a Multiple-Selection Window	B-18
B.6.7.1	Using the Chosen/Enable Item Attribute	B-18
B.6.7.2	Simulating Cursor Movement	B-19
B.6.7.3	Using the Don't Redisplay Current Items Attribute.....	B-19
B.6.7.4	Using the Displayed Attribute	B-20
B.6.7.5	First Item to Be Displayed	B-20
B.6.8	Using Windows for More Than One Purpose.....	B-21
B.6.9	Special Repaint on Receive	B-21
B.7	Designing Screen Files	B-22

Introduction

B.1 This appendix contains further explanations, examples, and hints on how to use the various Window Manager features. All of the example windows in this chapter can be created with Screen Builder. For practice you may want to try to recreate the windows.

Window Types

B.2 Window Manager works with six different types of windows:

- List
- Text
- Display
- Edit
- File
- User-defined

These six different types of windows can be used to display a wide variety of information. Suggestions on how to use list, text, display, edit and file windows are provided in the following paragraphs; user-defined windows are explained in Chapter 8, User-Defined Windows.

List Windows

B.2.1 The list window should be used whenever a list of items is presented from which selections will be made. It is the most frequently used window type. When designing user interfaces, you should utilize list windows as often as possible because it is much easier for a user to select items from a list than to type information. For example, earlier versions of Screen Builder prompted for attribute values by using edit windows. This process required memorization of all the different values. In the latest versions, the edit windows have been changed to list windows. This makes setting the attributes much easier and eliminates the need to memorize specific numbers.

Window Manager is very flexible in the way it can manipulate list windows. All the different formats and cursor types can be used with list windows. Window labels and item labels can also be used to display special information. When a Receive call is made on a list window, Window Manager can return to the application after one item is selected. If the Multiple Selection attribute is set to 1, Window Manager allows several items to be selected; then all selected items can be committed at the same time.

Text Windows **B.2.2** A text window is useful for displaying the results of a selection series from list windows. In such a case, there are no items to select, but there may be too many items to fit into the window at one time. A text window is perfect for this purpose because, while it does not allow a user to select items, the text can be scrolled. When the **arrow** keys are pressed, the text scrolls one line at a time. When the page scroll keys are pressed, the text scrolls one window at a time.

Text windows have the same appearance as list windows. All of the different formats, as well as window and item labels, can be used. An invisible cursor is used in text windows so that a user will not think that item selection is possible.

Display Windows **B.2.3** When information needs to be displayed but there is no reason for a user to manipulate window items in any way, a display window should be used. This type of window is used for display purposes only. The cursor cannot enter a display window, nor can a Receive call be made. Because display windows cannot be scrolled, they must be made large enough to contain all the items desired. Further information on display windows is provided later in this appendix.

Edit Windows **B.2.4** Edit windows enable users to type specific information. Edit windows can use all the different formats and can have window and item labels. Either a cursor or an invisible cursor is used in an edit window. Window Manager does not perform any data validation in edit windows. If validation is desired, the developer can provide an application validation routine which Window Manager will call. (See Chapter 7, Application Validation Routine, for more information.) Edit windows are more complex than other window types, since there are more attributes to set for items. These attributes determine how the items are displayed and manipulated.

Maximum Edit Field Length **B.2.4.1** An application typically has a maximum number of characters it allows a user to enter for an item. This maximum number is the value to which you set the Maximum Edit Field Length attribute. This item attribute tells Window Manager how many characters can be entered for the item. If this value is 0, the cursor can be placed on the item but no characters can be entered. Any item with a maximum length of 0 cannot be padded with fill characters. Blank lines (items) in edit windows can be created by specifying an item with no text and a maximum edit field length of 0.

Required Edit Field **B.2.4.2** The Required Edit field attribute can be utilized to force a user to enter characters for an item. When the attribute is set to 1, Window Manager does not allow the user to move the cursor to another item until at least one character has been entered for the item. This attribute is checked when the **ENTER**, **F10**, or **down arrow** keys are pressed. If an undefined key is pressed (one that will make Window Manager return from a Receive call), the required flag will be ignored.

Echo Edit Field Input **B.2.4.3** Sometimes an application program may require a user to enter sensitive information such as user IDs or passwords. Since these should not be seen, Window Manager can keep them confidential by using the Echo Edit Field Input attribute. When this attribute is set to 1, Window Manager displays the typed characters. If the value is 0, the cursor moves through the edit field but none of the typed characters appear.

Multiple Selection in an Edit Window **B.2.4.4** At times it is desirable for several edit items to be presented in a single window. The user can then enter information for all items and commit all of the entries at the same time using the Multiple Selection attribute.

When the Multiple Selection attribute is set to 1 for an edit window, Window Manager permits all fields in the window to be edited before returning to the application. Pressing the **ENTER** key moves the cursor to the next item. Pressing the **ENTER** key on the last item commits all edits. Pressing the **F10** key commits all edits at any time.

File Windows **B.2.5** There are many times when an application program has a text file that the user must view. Instead of the application reading the file and creating window items from the text, file windows can be used so Window Manager can perform the processing necessary to display the file in a window. All of the different formats can be used with file windows, but because of the nature of text files, the use of a single-column window is suggested. File windows can have window labels, but no item labels are allowed.

Window Manager reads the file and builds items from the text. The length of each item is determined by looking for carriage return characters in the file. When a carriage return is encountered, a new item will begin. If no carriage return characters exist in the file, the results of the file window cannot be guaranteed. Other restrictions also pertain to file windows. See Chapter 2, Window Manager Features, for more information on window types.

Window Labels

B.3 Window labels are typically used as titles of one or more lines (in uppercase letters as in Figure B-1) for a window. The window label can appear either flush left, centered, or flush right on the top line inside a window, or flush left and centered on the bottom line inside a window. The label remains in the position selected regardless of any scrolling within the windows. If the window label is invisible (that is, the Invisible Label attribute value is set to 1), it does not take up space in the window and it does not appear when the window is displayed. Figure B-1 shows four single-column list windows, each with the window label in a different position. Three of the windows shown have multiple-line labels. Even though only one window type and format is shown, all of the different window types and window formats can have window labels in each of the four positions.

Figure B-1

Window Label Positions

TOP WINDOW LABEL Item one Item two Item three Item four Item five Item six	LEFT WINDOW LABEL SECOND LINE OF LEFT LABEL Item one Item two Item three Item four Item five Item six Item seven
Item one Item two Item three Item four Item five BOTTOM WINDOW LABEL SECOND LINE OF BOTTOM LABEL	Item one Item two Item three Item four Item five Item six Item seven RIGHT WINDOW LABEL SECOND LINE OF RIGHT LABEL

Top Window Label

B.3.1 The top window label always appears on the first line or lines of the window. It can be centered or flush left. If the label is centered, it is centered over the entire width of the window. If a multiple-line window label is centered, each line is individually centered over the width of the window. This is true even if the window is either a single-column or multiple-column window. When using a top window label, the items will follow on subsequent lines. The first item(s) in a window that uses a top window label *cannot* appear on the same line(s) with that label. Items can be centered or flush left as shown in Figure B-1. Flush left is the default position for a top window label.

**Bottom
Window Label**

B.3.2 The bottom window label has the same characteristics as the top window label, except that it occupies the last lines in the window. The last items in a window that uses a bottom window label *cannot* appear on the same lines with that label. In Figure B-1, both the bottom label and the items are centered.

Left Window Label

B.3.3 The left window label always appears flush left on the first line(s) and in the first character position of the window. Window items are positioned immediately to the right of the left window label with no space separating the label and the first item. Unlike top and bottom window labels, no items can be displayed in the area beneath a left window label. Thus, usable window space is substantially reduced. Centered items are centered within this reduced area. If you do not wish to center items but do want space between the window label and the items, the label itself must be manually padded with the number of spaces desired. Padding with blanks allows great flexibility when designing the windows. Left or right window labels are typically used to label single-line windows or to provide the prompt in a single-line window.

Because of their reduced usable area, the column width of multiple-column windows is calculated by subtracting the width of the left or right window label from the overall width of the window, and dividing the result by the number of columns.

If a multiple-line left window label is used, each line appears flush left in the window, and the usable area of the window is based on the longest line in the window label. The Centering attribute has no effect on left window labels.

Right Window Label

B.3.4 The right window label has essentially the same characteristics as the left window label, except that it occupies the right side of the window on the top line(s). Items are positioned to the left of a right window label, and space below the label is unusable. Centered items are centered between the usable space to the left of the label.

If a multiple-line right window label is used, each line appears flush right in the window, and the usable area of the window is based on the longest line in the window label. The Centering attribute has no effect on right window labels.

Window Formats

B.4 Window Manager provides two formats in which the items can be displayed in a window: columnar and free-format. A columnar window can have either single-column or multiple-column items. Items are handled differently depending on the format selected and the size of the item. Figure B-2 illustrates four variations that occur using these two window formats. The four windows shown in this figure are list windows, but all window types can have these formats.

Figure B-2

Window Column Formats

SINGLE COLUMN WINDOW
Item one Item two Item three is too wide for one line. It has been made into a multi-line item.
MULTIPLE COLUMN WINDOW - ROW MAJOR ORDER
Item one Item two Item three Item four Item five Item six will be trun Item seven Item eight Item nine Item ten Item eleven
MULTIPLE COLUMN WINDOW - COLUMN MAJOR ORDER
Item one Item five Item nine Item two Item six will be trunItem ten Item three Item seven Item eleven Item four Item eight
FREE FORMAT WINDOW
This is item one. This is item two. Item three begins a new line, it didn't fit on previous line. Item four is wider than the width of the window; it will be tru The rest are all short items. Item six. Item seven. Item eight.

**Single-Column
Format**

B.4.1 Items in single-column windows are displayed one per line. The items can be either flush left or centered as shown in Figure B-2. Items wider than the width of the window are made into multiple-line items (for example, Item three in the single-column window of Figure B-2). Multiple-line items start at the left side of the window and are wrapped at the right side of the window. The next line of the item is indented two spaces. When an item-only size or larger-size cursor is placed on a multiple-line item, the cursor automatically expands in size to encompass the entire item.

If no multiple-line items are desired, the Disable Multiple Line Items attribute can be set to 1. When this is done, any item that is too long is truncated. Window Manager scrolls the items in windows that have the Disable Multiple Line Items attribute value set to 1 much faster than if the attribute value is 0.

**Multiple-Column
Format**

B.4.2 Multiple-column format is a simple variation of the single-column theme. Items are displayed in more than one column. The width of a column is determined by dividing the usable width of the window (window width is explained in Chapter 3, Window Manager Features) by the number of columns specified for the window. Normally, this would be the overall width of the window unless a left or right window label is used. The width of each column is the same. If the usable window width does not divide evenly between the columns, the extra width is left unused after the last column in the window.

The items can be left-justified as in the column-major window, or centered as in the row-major window. When multiple-column items are centered, they are centered over the width of the column. Items longer than the width of a column are truncated as shown by item six in both multiple-column windows of Figure B-2.

Window Manager does not put any spaces between the columns. This allows the screen designer maximum flexibility. If an item is as wide or wider than the width of the column, it runs into the next column. This is shown by Item six in the column-major window of Figure B-2. Two of the examples in Figure B-2 are multiple-column windows. They are the same except for the order in which the items are listed. In a *row-major* ordered window, the items are placed in the window one after another, filling up one row before moving down to start the next row.

In a *column-major* ordered window, the items are displayed in just the opposite order. The columns are filled first. The items are divided equally between the columns.

Cursor movement between items in a multiple-column window varies with the order in which the items are displayed. Left and right cursor movement in a row-major ordered window causes the cursor to move to the previous or next item in numeric order even if the item is on the previous or next line in the window. For example, if the cursor is on Item three in the row-major window of Figure B-2 and the **right arrow** key is pressed, the cursor moves to the next line and is positioned on Item four. Up and down cursor movement in a row-major window makes the cursor move only through the column in which it is positioned. It does not go to the previous or next column.

Cursor movement in a column-major ordered window is just the opposite of that in a row-major ordered window. In a column-major window, the up and down movement moves the cursor to the previous or next item. For example, if the cursor is on Item four in the column-major window of Figure B-2 and the **down arrow** key is pressed, the cursor moves to Item five in the next column. Left and right cursor movement can only be performed within a row. The cursor will not go to the previous or next row.

Free Format B.4.3 In a free format window, the items follow one after the other on the same line with one space placed between them. In Figure B-2, each item in the free format window ends with a period. If an item cannot fit on the same line following the previous item, the item wraps to the next line (see Item three). If a single item is longer than the width of the window, the item is truncated (as in Item four).

Cursor movement in a free-format window is very similar to cursor movement in a row-major ordered, multiple-column window. Left and right movement moves the cursor to the previous or next item. Up and down movement moves the cursor to the first item in the previous or next line. For example, if the cursor is on the third item in a line and the **up arrow** is pressed, the cursor moves to the first item in the previous line, even if there appeared to be an item directly above the item on which the cursor was positioned.

Item Labels

B.5 Item labels are typically used as prompts for edit items; however, they can be used in all window types except file windows. Window Manager associates the item label with the item text itself in determining how the window is displayed. Labels can be left- or right-justified, or free-format. Items, with or without labels, can be either centered or uncentered. Figure B-3 shows how a window appears using certain label characteristics in conjunction with specific item attributes.

Figure B-3

Item Labels

LEFT-JUSTIFIED ITEM LABELS - EDIT WINDOW
Uncentered Items **Centered Items**

Label 1: Item one____	Label 1: Item one____
Label 2: _____	Label 2: _____
Label 3: Item three will be trun	Label 3: Item three will be trun
Item four_	Item four_
Label 5: Item five_____	Label 5: Item five_____

RIGHT-JUSTIFIED ITEM LABELS - EDIT WINDOW
Uncentered Items **Centered Items**

Label 1:Item one____	Label 1:Item one____
Label 2:_____	Label 2:_____
Label 3:Item three will be trun	Label 3:Item three will be trun
Item four_	Item four_
Label 5:Item five_____	Label 5:Item five_____

FREE-FORMAT ITEM LABELS - LIST WINDOW
Uncentered Items **Centered Items**

Label 1:Item one	Label 1:Item one
Label 2:	Label 2:
Label 3:Item three will be a multiple	Label 3:Item three will be a multiple
line item.	line item.
Item four	Item four

Maximum Item Label Length

B.5.1 To display a label, you must specify the Maximum Item Label Length attribute. All justification takes place within this specified length. The item labels can be left or right justified, or free-format. If the maximum item label length is 0, no item labels appear even if text exists. This feature can be very useful if there is text you wish to store with the item but do not wish to display. The maximum item label length for the example windows is 13. If the maximum item label length is greater than the column width, Window Manager forces it to equal the column width.

Left-Justified Labels

B.5.2 The top window in Figure B-3 illustrates left-justified item labels. The item labels are left justified within the maximum item label length, and the items are positioned immediately following this maximum length. When no label text exists, as in Item four, the item is still positioned after the maximum item label length. This is useful because it allows all items in a window to be spaced over an equal distance in the window without having to pad items with blank spaces or specify item label text to obtain the proper spacing. If the label text is longer than the maximum length, the item label is truncated.

In multiple-column windows, free-format windows, edit windows, or windows with the Disable Multiple-Line Items attribute set to 1, the item is truncated first (before the label, as shown in Item three), if the item length plus the maximum item label length is wider than the width of the window. If multiple-line items are allowed, the item is word wrapped to the next line, indenting two spaces from the beginning of the item (as shown in the third item of the free-format example window in Figure B-3).

Items with left-justified labels are centered over the total length of the item text and the Maximum Item Label Length field.

Right-Justified Labels

B.5.3 Right-justified labels work in exactly the same manner as left-justified labels, except, of course, that the label is right justified within the Maximum Item Label Length field.

Free-Format Labels

B.5.4 With free-format item labels, the item immediately follows the item label itself. The maximum item label length is ignored for a free-format label, except if the item label text is longer than this maximum length. In this case, the label is truncated to fit the maximum length. If no label text exists for the item, the item label is ignored (as shown in Item 4 of Figure B-3). A free-format item label and an item are centered over the length of the item and the actual length of the label text.

Special Window Manager Features

B.6 This section discusses some of the more complicated window attributes and how these features can be used to create special effects not described elsewhere in the manual. Even though each of the following paragraphs focus on a different topic, it is recommended that the whole section be read through carefully to obtain a good overall understanding of how Window Manager functions.

Window Manager has many features which allow special window effects to be created. Figure B-4 illustrates these features. The Window Label Attributes menu is used to set the window label attributes for the NaturalLink Screen Builder utility. Refer to this figure as you read the following paragraphs in order to better understand the concepts presented here.

In the following discussion, Figure B-4 is referred to as a screen even though it does not take up the entire display area.

Figure B-4

Window Label Attributes

```
Set Window Label Attributes

Invisible label :   YES   NO
Label position :   [ ]   LEFT   RIGHT   BOTTOM
Centered label :   YES   [ ]
Use item intensity : YES   [ ]
Intensity/color :  0/BLACK  1/BLUE  2/RED  3/MAGENTA
                  4/GREEN  5/CYAN  6/YELLOW  7/WHITE
Underlined :      YES   [ ]
Blinking :        YES   [ ]
Reverse video :   YES   [ ]

Press the F10 key to commit your selections
```

Use of Borderless Windows and Multiple Window Effects

B.6.1 The example screen in Figure B-4 is actually three windows made to appear as one. This Window Manager capability is a very powerful aid in displaying information. This effect is created by placing two borderless windows on top of a larger, bordered window. The double-ruled lines in Figure B-5 indicate where the invisible borders are.

Figure B-5

Invisible Borders

Set Window Label Attributes				
Invisible label :	YES	NO		
Label position :	<input type="checkbox"/>	LEFT	RIGHT	BOTTOM
Centered label :	YES	<input type="checkbox"/>		
Use item intensity :	YES	<input type="checkbox"/>		
Intensity/color :	0/BLACK	1/BLUE	2/RED	3/MAGENTA
	4/GREEN	5/CYAN	6/YELLOW	<input type="checkbox"/>
Underlined :	YES	<input type="checkbox"/>		
Blinking :	YES	<input type="checkbox"/>		
Reverse video :	YES	<input type="checkbox"/>		

Press the F10 key to commit your selections

One of the two borderless windows contains the prompts (Invisible label, Label position, and so on), while the other contains the items to be selected (the Yes or No items). In the following discussion, these windows are referred to as the *prompt window* and the *item window*, respectively. The large bordered window, which encompasses the prompt and item windows, is referred to as the *label window*.

Adding Multiple Windows in Order

B.6.1.1 The order in which these three windows are displayed is very important. The label window must be added first, followed by the prompt window, and then the item window. If this order is not followed, the label window would cover the two smaller windows.

This applies not only to the use of multiple-window effects, but to the use of windows in general. The order in which windows are added (by making a Window Manager Add Window call) is the order in which the windows are displayed by Window Manager.

Reasons for Using Multiple Windows

B.6.1.2 Multiple windows can be used to achieve the following effects:

- The simultaneous display of a variety of information.
- Combined flexibility and greater functionality of several window formats with the simplicity of single-window use.
- Enhanced menu appearance.

Some background information regarding the design of the Set Window Label Attributes screen may be helpful in explaining these points and is presented in the following paragraphs.

The purpose of the Set Window Label Attributes screen is to present prompts for window attributes and allow a user to select the answers to these prompts. A title is required for the screen as well as information telling the user what to do after making all of the selections.

One multiple-column window is ineffective for presenting all this information because the column width for each column in a multiple-column window must be the same. Long prompts cannot fit in the area available in such a window. In addition, one multiple-column window can only have one window label. At times it is necessary to have a title at the top as well as an information line at the bottom. This can be done using a single multiple-column window, but not easily.

In one free-format window, the items can be spaced to give the impression of a multiple-column window. However, cursor movement in a free-format window is not the same as in a multiple-column window. Thus, a user may become confused because the window appears to be a multiple-column window. Another drawback is that spacing items in a free-format window takes time and is not easy to update. If possible, it is always best to design windows so that Window Manager handles all item formatting.

Therefore, three windows are used to display the screen.

- A single-column window used to display the prompts. The prompts can be any size and are not tied to the column width of a multiple-column window.
- A four-column window used to display the items to be chosen.
- A large single-column window, which provides the border as well as the title and information line desired, to encompass the other two windows.

**Use of
Display
Windows**

B.6.2 Another reason to use three separate windows concerns cursor movement. The cursor should be placed only on selectable items. The cursor should not be able to move all over the screen. This is easily controlled by the use of display type windows for the label and prompt windows. The cursor can never enter display windows, and items inside the windows cannot be selected. Therefore, you can have unselectable items without having to set the Unselectable Item attribute for every prompt.

Creating and Using Blank Lines

B.6.3 Spacing plays a very important role in making the information look good and in making it easier to read. Spacing is easily controlled by the use of blank lines (items) in the window. For example, the blank spaces between the Intensity/color and Underlined prompts, as well as the blank areas in the items window, are all created by using blank items.

These blank items are created by defining an item but not specifying any text for this item. Spaces only need to be specified for blank lines in free-format windows. In a columnar format, Window Manager leaves a blank space for an item that has no text specified.

As previously discussed, the blank lines in the items window are actually items without any text. These are needed because the window is a multiple-column window and, in order to reserve space, an item must be present. Thus, the lines that appear to have just Yes and No in them actually have two values and two blank items. This provides the spacing needed to make the lines look as if some prompts have two choices while others have four or more.

When using blank lines, you should consider whether there are enough items in the window so that the items can be scrolled; blank lines are scrolled along with any other items.

Unselectable Items

B.6.4 Blank items are commonly used in conjunction with unselectable items. Window Manager never positions the cursor on items that have the Unselectable Item attribute set to 1. The blank items used to space the multiple-column item window have the Unselectable Item attribute set to 1; therefore, the cursor is positioned only on items that should be selected.

Cursor movement in a multiple-column window with unselectable items may work quite differently than expected. When an **up** or **down arrow** key is pressed, the cursor moves to the previous or next selectable item, respectively, in that same column. If the window is in column-major order and no selectable item is found, the cursor goes to the next column. A similar movement occurs for left and right movement within a row. (The cursor goes on to the next row in a row-major ordered window.) In Figure B-5, if the cursor is on the RIGHT item and a **down arrow** is pressed, the cursor moves to the 2/RED item, skipping over the unselectable items in between.

**Use of
Invisible Items**

B.6.5 Making items invisible (by setting the Visible Item attribute to 0) has great advantages. If items in a window should be visible some of the time, define those items initially and then set the Visible Item attribute on and off as needed. This option is far superior to creating and deleting items.

In Figure B-4, invisible items are used extensively. Initially, all of the items used in the Set Window Label Attributes screen are shown. However, if Yes is selected in response to the Use Item Intensity prompt, all items dealing with Intensity/color disappear. This is accomplished by setting the Visible Item attribute for these items to 0 and redisplaying the window.

If you decide to change the window label attributes and select No in response to the Use Item Intensity prompt, the Visible Item attribute is used in the opposite manner to bring the items back into view. The invisible items have their Visible Item attribute set to 1 and the window is redisplayed.

After setting the Visible Item attribute, the window must be redisplayed by making either a Display or a Refresh call followed by a Receive call. This is because the attribute changes do not show up on the display until the window is repainted again. This is true not only for the Visible Item attribute but for any attribute set.

**When to Use
Multiple-Selection
Windows**

B.6.6 Multiple-selection windows are very helpful when a user needs to select several items of the same type. For example, in the Draw Screen command of the Screen Builder utility, if a subset of the windows is desired, the window from which the user is making a selection is a multiple-selection window. When each item is selected, Window Manager sets the Chosen/Enable attribute to 1. When this attribute is set to 1, the item is displayed using the Item Chosen attributes. Selecting an item which was previously selected results in the Chosen/Enable attribute being toggled. In other words, if an item has been selected, it is unselected if chosen again.

**Simulating a
Multiple-Selection
Window**

B.6.7 Sometimes a multiple-selection window is not a good choice because of the background tasks necessary in this type window, as is the case for the Set Window Label Attributes screen. The effects that a multiple-selection window provides (that is, the toggling of selected items and the display of selected items, using the Item Chosen attributes) are desired, but the fact that a user stays in a multiple-selection window until the F10 key is pressed is not desired. The Screen Builder has to know what is chosen, so it can make certain items visible or invisible after each selection. Thus, other Window Manager features can be used to simulate a multiple-selection window. The ways to accomplish this process are described in the following paragraphs.

**Using the
Chosen/Enable
Item Attribute**

B.6.7.1 To simulate a multiple-selection window, the selected items must be displayed in a different manner than the unselected items. This is done by setting the Chosen/Enable Item attribute. When this attribute value is 1, Window Manager displays the item using the values of the Item Chosen attributes. When set to 0, the item is displayed using active or inactive window attributes (depending on the state of the window). Window Manager sets the Chosen/Enable Item attribute and redisplay the item automatically in multiple-selection windows; the same effect can be achieved by setting the Chosen/Enable attribute and redisplaying the item manually in single-selection windows.

In Figure B-4, the Item Chosen attribute values are set to the default values of Reverse video (1) and Intensity/color (7/WHITE).

When an item is selected (for example, No), its Chosen/Enable Item attribute is set to 1; the Chosen/Enable Item attribute of its counterpart (Yes) is set to 0, and the window is redisplayed. Thus the chosen item is displayed in reverse video and the unselected item is displayed using the active attributes. For example, when Yes is selected for the Centered Label prompt, the Chosen/Enable Item attribute for the Yes item is set to 1 and the Chosen/Enable Item attribute for the No item is set to 0. Thus Yes is in reverse video and No is not.

The Chosen/Enable Item attribute can be used whenever an item is to be displayed with different attributes from other items in the window. Remember, to display any attribute changes, the window must be redisplayed by making a Display call or by making a Refresh call followed by a Receive call. In Figure B-4, the items shown in reverse video gray indicate items with the Chosen/Enable attribute set.

***Simulating
Cursor Movement***

B.6.7.2 In a multiple-selection window, the cursor automatically goes to the next item in the window once an item is selected. This can easily be simulated by specifying the item on which you wish the cursor to be positioned in the Receive call. Window Manager always tries to place the cursor on the item passed on the Receive call. If the item number passed is either out of range or not a valid item (either unselectable or invisible), Window Manager tries to return the cursor to its previous position. If this position does not contain a valid item, the cursor is placed on the first valid item displayed in the window.

In Figure B-4, every time a Receive call is executed on the Items window, the item that the cursor should be placed on is explicitly specified. After a label position has been selected, the next Receive call is made by passing in either the Yes or No item for the Centered Label prompt.

***Using the Don't
Redisplay Current
Items Attribute***

B.6.7.3 Redisplaying a window just to change the appearance of one or two items in that window causes a flashing effect on the video display. This is because all the items are being repainted as well as the ones which are changing. This unnecessary repainting of all items and excessive screen flash can be controlled by using both a window attribute and an item attribute.

The window attribute that must be used is the Don't Redisplay Current Items attribute. When it has a value of 1, only items which have not been displayed yet are displayed. This eliminates the flash caused by every item in the window being repainted. This attribute can be used by itself for situations in which items are added to the end of a list of items in a window during processing. When the window is repainted, only those new items added to the end of the list are displayed. Window Manager does not redisplay the items currently in the window.

Care must be taken when using this attribute because Window Manager assumes that the order of the items has not changed since the last time the window was painted. If the order of items has been changed, the entire window must be repainted. To accomplish this when the Don't Redisplay Current Items attribute is set, a different First Item to Be Displayed attribute must be set for the window. A value of -1 guarantees that the entire window is repainted.

***Using the
Displayed
Attribute***

B.6.7.4 The Don't Redisplay Current Items attribute is most powerful when used with the Displayed Item attribute. The Displayed Item attribute is set to 1 by Window Manager after the item is displayed. Thus when the Don't Redisplay Current Items attribute is set to 1, Window Manager only displays those items whose Displayed attribute has a value of 0.

Screen Builder uses this Displayed attribute in all screens similar to the example. Both the prompt window and the items window have the Don't Redisplay Current Items attribute set to 1. Whenever an item is selected (for example, No), Screen Builder completes the following:

- Sets the Chosen/Enable attribute for the selected item.
- Turns off the Chosen/Enable attribute for the corresponding item (the Yes item).
- Sets the Displayed Item attribute for both items to 0.
- Redisplays the window. This way, only the items whose attributes change are redisplayed, not the entire window.

It is important to note that when the Don't Redisplay Current Items attribute is set, Window Manager will display only the items that have not yet been displayed. This is true no matter which Window Manager call is made. The only way to have Window Manager redisplay all of the items is either to change the value of the First Item to Be Displayed attribute or to set the Displayed attribute of all items to 0.

***First Item
to Be Displayed***

B.6.7.5 The First Item to Be Displayed attribute can be used to determine the first item Window Manager displays in a window. It can be set to any item in the window, and Window Manager starts displaying items from this point the next time the window is displayed.

This attribute also has a very important use in windows that have the Don't Redisplay Current Items attribute set to 1. Because Window Manager does not redisplay all of the items, there must be some way to signal Window Manager that the entire window should be repainted. This is accomplished by setting the First Item to Be Displayed attribute to a value different from what is currently set. It is recommended that a value of -1 be used. If the First Item to Be Displayed is valued -1, Window Manager starts displaying the window with the first visible item in the item list.

Screen Builder uses the First Item to Be Displayed attribute when it has made some items invisible in the Set Window Label attributes screen and wants to make them visible again. Just setting the Displayed attribute to 0 causes those items to be repainted but does not reorganize all of the items. If this is done with Intensity/color items, these items cover up any items currently showing at that spot in the window. However when the First Item to Be Displayed attribute is set to -1, Window Manager knows to repaint the entire window, and the Intensity/color items fall into place.

**Using Windows
for More Than
One Purpose**

B.6.8 So far in this discussion, only prompt and item windows have been considered. A discussion of some of the design issues relating to the large label window follows. The label window is used as the background window for prompt and item windows. In Figure B-4, a bottom window label is used as the instruction line and an actual item is used for the title line. This was done because placing an item at the bottom of the window would be more difficult, requiring enough items to be defined to space the desired item out to the bottom line.

The set of three windows in Figure B-4 is used in more than one place in the Screen Builder utility. The screens to set active, inactive, window label, and cursor attributes are all similar except for a few items. Therefore, the same three windows can be used and different items can be made visible depending on how the windows are being used.

It is necessary to have different titles in various locations in the Screen Builder utility label window. For this reason, an item is used for the title in the label window. Four items are actually defined for this large window, and a different one is made visible for each place that this window is used. Four different titles could not be easily produced by using the window label since a window can have only one window label at a time.

**Special Repaint
on Receive**

B.6.9 The best way to explain this attribute is by example. When setting the window format in Screen Builder, you are presented with a large screen of choices. This screen is actually comprised of three windows in the same way as Figure B-4. However, when setting window formats, there are times when small pop-up windows appear in which you must type specific values. Once the value is entered, windows covered by the pop-up window are repainted.

All of these functions are easily performed by making Window Manager calls. Window Manager handles the display of any windows marked for repainting the next time a Receive call is made. When the pop-up window is deleted, the covered windows are marked for repainting. In the Set Window Format screen, this means that all three of the windows are marked for repainting.

When the next Receive call is executed on the items window, all three of these windows are redisplayed. This redisplay of all windows produces an undesirable effect. The large label window takes a long time to redraw, and since its only vital information (the title and the bottom line) are not covered by the pop-up screen, it does not need to be repainted.

This is why the Special Repaint on Receive attribute is useful. This attribute can be set to 1 for the items window. When the Receive call is executed on the items window, only that window is repainted. Although the prompt window is also covered by the pop-up window, the prompt window can be explicitly repainted by making a Display call. This results in the screen being repainted quickly with only the information that needs to be redisplayed.

Designing Screen Files

B.7 When using Screen Builder to design and create screens for an application program, you should follow a few guidelines.

It is best to put logically related windows in the same screen file. If a series of windows is used in the same place, these windows are likely candidates for placement in the same screen file. It is undesirable to have one window per file because loading the screen into memory takes longer than almost any of the other Window Manager functions.

Having too many windows per screen file can be as problematic as having too few. A screen file with more than twenty windows is hard to manage. Typically, such a screen file can be split into one or more logically related units.

A good working screen file is one that contains all windows for one display as well as any related pop-up windows. This way the screen files are easy to keep track of and can be named according to where they are used in the application program.

COMPUTER-SPECIFIC INFORMATION

C

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
C.1	Introduction	C-3
C.2	Computers Supported	C-3
C.2.1	List of Computers	C-3
C.2.2	Machine Detection	C-3
C.3	TI Computers	C-4
C.3.1	TI BUSINESS-PRO Computer	C-4
C.3.2	TIPC/TIPPC Function Key Differences	C-5
C.3.3	TIPC Character Codes	C-5
C.3.4	TI PRO-LITE Computer	C-9
C.3.4.1	Considerations for TI PRO-LITE Computer	C-11
C.3.4.2	LCD Flag	C-12
C.4	IBM PC, PC/XT, and Personal Computer AT	C-12
C.4.1	Underlining	C-12
C.4.2	Reverse Video	C-13
C.4.3	Intensity/Color	C-13
C.4.4	Cursor Attributes	C-13
C.4.5	Key Codes	C-13

Introduction

C.1 This appendix describes the different computers on which the NaturalLink Window Manager software can run. It also describes any functionality differences between the computers, such as video attributes, function keys, and so on.

Computers Supported

C.2 The NaturalLink software has been tested and verified to run on the following computers.

Texas Instruments

BUSINESS-PRO Computer
Professional Computer
Portable Professional Computer
PRO-LITE Professional Computer

IBM

PC
PC/XT
Personal Computer AT

List of Computers

C.2.1 The use of TIPC in this manual refers to the Texas Instruments Professional Computer family, which includes the TI BUSINESS-PRO Computer (TI mode), TI Professional Computers (TIPC and TIPPC), and TI PRO-LITE Computer. The use of the phrase "other computers" refers to the IBM family of computers, including the IBM PC, PC/XT, and Personal Computer AT, and to the TI BUSINESS-PRO Computer (PC-AT mode).

Machine Detection

C.2.2 A flag called Llpctype (pctype for FORTRAN users) has been provided and can be used to determine whether your NaturalLink application is running on a TI or IBM computer. Llpctype is set by the first WMINIT call and has one of the following values once it is set.

<i>Value</i>	<i>Definition</i>
10	TIPC/TIPPC
11	TI PRO-LITE
12	TI BUSINESS-PRO (TI mode)
20	IBM PC
21	IBM PC/XT
22	IBM Personal Computer AT
23	TI BUSINESS-PRO (PC-AT mode)

Llpctype is set only once, no matter how many WMINIT calls are made. It is a read-only attribute, meaning that a WMSETV call cannot be used to set it.

**TIPC/TIPPC
Function Key
Differences**

C.3.2 All the references to function keys in the manual relate to the TI BUSINESS-PRO Computer running in PC-AT mode. There are some differences in the keys used when running on the TIPC, TIPPC, and TI PRO-LITE Computer. Table C-1 indicates these differences:

Table C-1

TI Computer Function Key Differences

<i>Documented Keys</i>	<i>Use on TIPC/TIPPC/ TI PRO-LITE</i>
ENTER	RETURN
CTRL-PgUp	CTRL-Up Arrow
CTRL-PgDn	CTRL-Down Arrow
CTRL-PrtSc	PRNT

The **ENTER** key can be mapped to be different from the **RETURN** key by assigning a code of 190H to any function, using KBUILD. This mapping is in effect between the time the Initialize (WMINIT) procedure and the Reset (WMRSET) procedure are called. When the **ENTER** key is mapped, the **SHIFT-RETURN** key combination is also mapped to return a code of 190H in Window Manager.

**TIPC
Character
Codes**

C.3.3 The key codes for the TI Professional Computer are documented in Table C-1. A TIPC United States keyboard layout is shown in Figure C-2. For an in-depth discussion of the keys, consult the *Texas Instruments Professional Computer Technical Reference Manual*. Following are some general notes for reference when you are using the table:

1. In each column, both the graphic and the hexadecimal values of the characters are given in the form: ggg hh.
2. Entries consisting of one long dash followed by one short dash (— —) indicate that the combination is suppressed within the keyboard Device Service Routine (DSR). These keys have no effect on Window Manager; that is, Window Manager does *not* return to the application program when any of these keys are pressed.

3. Entries consisting of xxx ** indicate special handling in the form of direct action by the keyboard DSR.
 - a. **BRK/PAUS** — The DSR performs the pause function unconditionally. This does not adversely affect Window Manager. When another key is pressed, Window Manager resumes normal operation. Window Manager does not return to the application program when **PAUS** is pressed.
 - b. **SHIFT-PRNT** — The SHIFT-PRNT (print screen) DSR function is performed unconditionally. This does not adversely affect Window Manager. Once the print screen function has finished, Window Manager resumes normal operation. Window Manager does not return to the application program when the **SHIFT-PRNT** keys are pressed.
 - c. **SHIFT-BRK/PAUS** — Window Manager returns to the application program. The key code returned is 100H. If the application then makes any call related to the operating system, the DSR program break function is performed.

The application can redirect the system interrupts to perform the application's own version of these three functions. Window Manager still handles the keys in the same manner, but the application's version of the function is performed instead of the normal system function.

4. Entries consisting of xxx yy * indicate that this key returns the extended code yy. If such a key is pressed and it is unknown to Window Manager (that is, does not invoke a Window Manager function), Window Manager returns this extended code, plus 100H, to the application program.
5. All other entries are normal ASCII keys, and if they are unknown to Window Manager, the exact normal code is returned.

Table C-2 Standard TIPC U.S. Keyboard Character Codes

<i>Scan Code</i>	<i>Normal</i>		<i>SHIFT</i>		<i>CTRL</i>		<i>ALT</i>		<i>Comments</i>
01	f5	3F★	sf5	58★	cf5	62★	af5	6C★	F5
02	f6	40★	sf6	59★	cf6	63★	af6	6D★	F6
03	f7	41★	f7	5A★	cf7	64★	af7	6E★	F7
04	f8	42★	sf8	5B★	cf8	65★	af8	6F★	F8
05	f9	43★	sf9	5C★	cf9	66★	af9	70★	F9
06	f10	44★	sf10	5D★	cf10	67★	af10	71★	F10
07	f11	45★	sf11	08★	cf11	0A★	af11	0C★	F11
08	f12	46★	sf12	09★	cf12	0B★	af12	0D★	F12
09	1	31	!	21	—	—	alt1	78★	
10	2	32	@	40	Fnul	03★	alt2	79★	
11	3	33	#	23	—	—	alt3	7A★	
12	4	34	\$	24	—	—	alt4	7B★	
13	5	35	%	25	—	—	alt5	7C★	
14	6	36		5E	RS	1E	alt6	7D★	
15	7	37	&	26	—	—	alt7	7E★	
16	8	38	*	2A	—	—	alt8	7F★	
17	9	39	(28	—	—	alt9	80★	
18	0	30)	29	—	—	alt0	81★	
19	—	2D	—	5F	US	1F	alt-	82★	
20	=	3D	+	2B	—	—	alt=	83★	
21	BS	08	BS	08	DEL	7F	—	—	Backspace
22	'	60	~	7E	—	—	—	—	
23	=	3D	=	3D	=	3D	pf1	8C★	Numeric =
24	+	2B	+	2B	+	2B	pf2	8D★	Numeric +
25	SP	20	SP	20	SP	20	pf3	8E★	Numeric
26	HT	09	Bktab	0F★	HT	09	pf4	8F★	SPACE
27	1	31	1	31	1	31	—	—	Numeric TAB
28	—	—	—	—	—	—	—	—	Numeric 1
29	0	30	0	30	0	30	—	—	(Unused)
30	CR	0D	CR	0D	CR	0D	—	—	Numeric 0
31	4	34	4	34	4	34	—	—	Numeric ENTER
32	5	35	5	35	5	35	—	—	Numeric 4
33	9	39	9	39	9	39	—	—	Numeric 5
34	—	2D	—	2D	—	2D	—	—	Numeric 9
35	2	32	2	32	2	32	—	—	Numeric -
36	—	—	—	—	—	—	—	—	Numeric 2
37	—	—	—	—	—	—	—	—	(Unused)
38	—	—	—	—	—	—	—	—	(Unused)
39	7	37	7	37	7	37	—	—	(Unused)
40	8	38	8	38	8	38	—	—	Numeric 7
41	6	36	6	36	6	36	—	—	Numeric 8
42	,	2C	,	2C	,	2C	—	—	Numeric 6
43	3	33	3	33	3	33	—	—	Numeric ,
44	.	2E	.	2E	.	2E	—	—	Numeric 3
45	—	—	—	—	—	—	—	—	Numeric .
									(Unused)

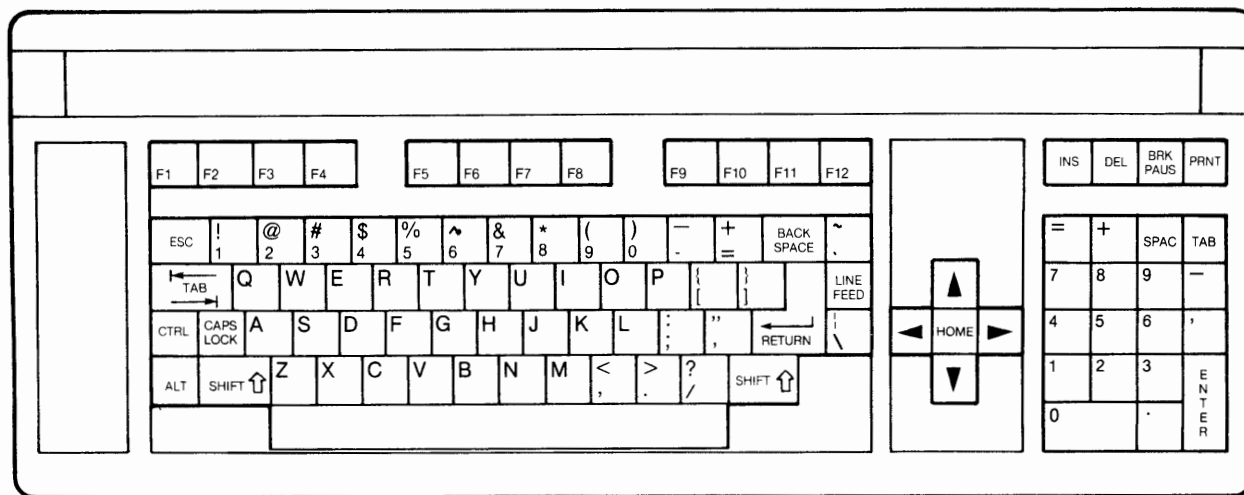
Table C-2 Standard TIPC U.S. Keyboard Character Codes (Continued)

<i>Scan Code</i>	<i>Normal</i>		<i>SHIFT</i>		<i>CTRL</i>		<i>ALT</i>		<i>Comments</i>
46	C-rt	4D★	sC-rt	8A★	cC-rt	74★	aC-rt	4E★	Right arrow
47	Ins	52★	sIns	28★	cIns	29★	aIns	2A★	INS
48	Del	53★	sDel	38★	cDel	39★	aDel	3A★	DEL
49	HT	09	Bktab	0F★	HT	09	—	—	TAB
50	q	71	Q	51	DC1	11	altQ	10★	
51	w	77	W	57	ETB	17	altW	11★	
52	e	65	E	45	ENQ	05	altE	12★	
53	r	72	R	52	DC2	12	altR	13★	
54	t	74	T	54	DC4	14	altT	14★	
55	y	79	Y	59	EM	19	altY	15★	
56	u	75	U	55	NAK	15	altU	16★	
57	i	69	I	49	HT	09	altI	17★	
58	o	6F	O	4F	SI	0F	altO	18★	
59	p	70	P	50	DLE	10	altP	19★	
60	[5B	{	7B	ESC	1B	—	—	
61]	5D	}	7D	GS	1D	—	—	
62	LF	0A	LF	0A	cLF	75★	aLF	4F★	Line feed
63	—	—	—	—	—	—	—	—	(Unused)
64	C-up	48★	sC-up	88★	cC-up	84★	aC-up	49★	Up arrow
65	ESC	1B	ESC	1B	ESC	1B	—	—	ESC
66	a	61	A	41	S0H	01	altA	1E★	
67	s	73	S	53	DC3	13	altS	1F★	
68	d	64	D	44	EOT	04	altD	20★	
69	f	66	F	46	ACK	06	altF	21★	
70	g	67	G	47	BEL	07	altG	22★	
71	h	68	H	48	BS	08	altH	23★	
72	j	6A	J	4A	LF	0A	altJ	24★	
73	k	6B	K	4B	VT	0B	altK	25★	
74	l	6C	L	4C	FF	0C	altL	26★	
75	;	3B	:	3A	—	—	—	—	
76	'	27	"	22	—	—	—	—	
77	CR	0D	CR	0D	CR	0D	—	—	
78	\	5C		7C	FS	1C	—	—	
79	C-lf	4B★	sC-lf	8B★	cC-lf	73★	aC-lf	4C★	Left arrow
80	Home	47★	sHome	86★	cHome	77★	aHome	85★	HOME
81	SP	20	SP	20	SP	20	SP	20	Space bar
82	z	7A	Z	5A	SUB	1A	altZ	2C★	
83	x	78	X	58	CAN	18	altX	2D★	
84	c	63	C	43	ETX	03	altC	2E★	
85	v	76	V	56	SYN	16	altV	2F★	
86	b	62	B	42	STX	02	altB	30★	
87	n	6E	N	4E	SO	0E	altN	31★	
88	m	6D	M	4D	CR	0D	altM	32★	
89	,	2C	<	3C	—	—	—	—	
90	Ptogl	72★	***	**	—	—	—	—	PRINT

Table C-2 Standard TIPC U.S. Keyboard Character Codes (Continued)

Scan Code	Normal		SHIFT		CTRL		ALT		Comments
91	.	2E	>	3E	—	—	—	—	
92	/	2F	?	3F	—	—	—	—	
93	—	—	—	—	—	—	—	—	(Unused)
94	—	—	—	—	—	—	—	—	(Unused)
95	—	—	—	—	—	—	—	—	(Unused)
96	C-dn	50*	sC-dn	89*	cC-dn	76*	aC-dn	51*	Down arrow
97	—	—	—	—	—	—	—	—	(Unused)
98	—	—	—	—	—	—	—	—	(Unused)
99	—	—	—	—	—	—	—	—	(Unused)
100	Ppau	**	Pbrk	**	—	—	—	—	BRK/PAUS
101	f1	3B*	sf1	54*	cf1	5E*	af1	68*	F1
102	f2	3C*	sf2	55*	cf2	5F*	af2	69*	F2
103	f3	3D*	sf3	56*	cf3	60*	af3	6A*	F3
104	f4	3E*	sf4	57*	cf4	61*	af4	6B*	F4

Figure C-2 TIPC U.S. Keyboard Layout

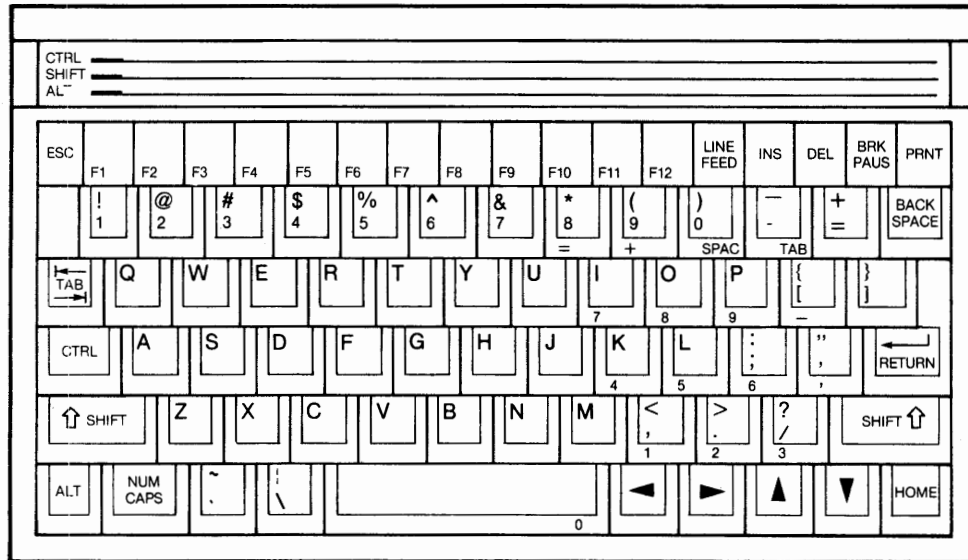


TIPRO-LITE Computer

C.3.4 A keyboard layout for the TI PRO-LITE Computer is shown in Figure C-3. Certain considerations must be taken into account when a NaturalLink application is run on the TI PRO-LITE Computer. Only applications linked with Version 1.75 or greater of the NaturalLink run-time object code will run correctly. Applications linked with earlier versions will run, but there will be no distinction between active and inactive windows on the liquid crystal display (LCD).

Figure C-3

TI PRO-LITE Computer U.S. Keyboard



The TI PRO-LITE Computer can use either the built-in LCD or an external monitor, but not both at once. Support for the LCD is automatic, as the NaturalLink software makes all checks necessary to function properly. When an external monitor is being used, NaturalLink functions the same as it does on the TIPC.

When the LCD is used, the following is **forced** to happen:

- Reverse video is used for the following:
 - All borders
 - Items in active windows
 - Window labels in active windows
 - Item labels in active windows
- The cursor in active windows is black (not reverse video)
- The chosen item attribute of reverse video, if used in active windows, is turned off

All other attributes remain the same, with the exception that all intensities are equal on an LCD.

**Considerations for
TI PRO-LITE
Computer**

C.3.4.1 When designing a NaturalLink application that will be run on the TI PRO-LITE Computer, you must keep several facts in mind to ensure that the application runs and looks good.

Inactive Windows Inactive windows must not have the Reverse Video attribute set.

Multiple-Window Displays Multiple-window displays whose combined effect is to look like one window must have the Window Manager Select and Release operations executed on all windows involved. For example, in Screen Builder, the screens used to set attributes are actually comprised of three windows: (1) a large background display window, (2) a display window on the left side, and (3) the window on which receive is executed. To achieve the one-window look, all three windows must be made active. This ensures that all three will be in reverse video on the TI PRO-LITE Computer.

Special Repaint Attribute In the Set Window Format option of Screen Builder, there are three windows comprising the screen. At various points a pop-up window is used to get specific information (number of columns, and so on). When the pop-up is deleted, an explicit Display call is made on the left window and a Receive call is made on the right window. The receive window has the Special Repaint attribute set, which means that the big background window is not repainted following deletion of the pop-up. This results in a black line where the two windows on top meet. Because the two top windows are unbordered, the border is ignored when they are repainted. Since the background window is not repainted, the black line is left for the border in the spot where the pop-up used to be. In situations like this, either make an explicit Display call on the large background window following deletion of the pop-up, or turn off the Special Repaint attribute.

Highlighted Items Items which are currently highlighted by using the Chosen/Enable Intensity attribute may not look right when the window is active. Since NaturalLink does not do anything with Chosen/Enable Intensity on the TI PRO-LITE Computer LCD, the item displays in normal video while the rest of the window is reverse video when the window is active. This results in the one item sticking out more than may have been desired.

SHIFT-RETURN and ENTER Keys Both the **SHIFT-RETURN** and the **ENTER** keys act the same as on the TIPC; they are mapped to return an extended code of 190H if any function has been assigned the value of 190H using KBUILD.

LCD Flag C.3.4.2 A flag called Lllcdon (lcdon for FORTRAN users) has been provided to signal NaturalLink that it is running on a liquid crystal display. This flag is set by the WMINIT call when Window Manager detects that it is running on a TI PRO-LITE Computer with an LCD. Lllcdon is set only once, no matter how many WMINIT calls are made. The application may read the value of Lllcdon by making a WMGETV call. A value of 1 indicates Window Manager is running on an LCD, while a value of 0 indicates a normal display.

Lllcdon may be used by an application for debugging purposes to fool the NaturalLink software into thinking it is running on a TI PRO-LITE Computer LCD. This is useful if an application is being developed for the TI PRO-LITE Computer when TI PRO-LITE Computer hardware is not available. The flag is set using the WMSETV call and, if used, must be set *after* the first WMINIT call has been made. When Lllcdon has a value of 1, NaturalLink thinks it is running on an LCD and displays any windows in the same way it would on the TI PRO-LITE Computer LCD.

IBM PC, PC/XT, and Personal Computer AT

C.4 Window Manager software was originally designed for use on the TI Professional Computer. Therefore, certain video attributes used by Window Manager are hardware-specific to TI equipment. The differences between the IBM computers and the TI computers are relatively minor, having to do only with the following:

- Underlining
- Reverse video
- Intensity/color
- Cursor attributes

Underlining C.4.1 There are two situations in which underlining is different on an IBM computer and on a TI computer.

1. When underlining and reverse video are set to function simultaneously, the Reverse Video attribute overrides underlining
2. IBM computers do not support underlining on a color monitor.

Reverse Video C.4.2 IBM computers support only one intensity level of reverse video on a monochrome monitor. When an item is displayed in reverse video on an IBM computer with a monochrome monitor, it is displayed in low intensity, regardless of which Intensity/color attribute has been set in the window description.

Intensity/Color C.4.3 The colors used on TI computers are the same as the colors for IBM computers. Differences in this attribute appear when a monochrome monitor is used. IBM supports only two levels of intensity on its monochrome monitor. Thus, if an item's Intensity attribute is set in the range of 0 through 4, the item appears in low intensity on an IBM computer. A range of 5 through 7 produces high intensity.

Cursor Attributes C.4.4 A single-character cursor on the IBM computers will always blink, regardless of how the Blinking Cursor attribute is set. Also, since underlining is not possible on a color monitor, full-size, item-and-label, and item-only cursors cannot be just an underline on an IBM color monitor.

Key Codes C.4.5 When running on an IBM computer, Window Manager returns key codes to the application in the same way as it does for TI computers. If the key is a normal ASCII character (that is, the key code is returned in register AL with the key number in AH), Window Manager returns this normal code. If the key is a special function key (non-ASCII code), which returns an extended code (that is, register AL = 0 with the extended code in AH), Window Manager returns the extended code plus 100H.

A keyboard layout for the IBM PC and PC/XT is shown in Figure C-4; a layout for the IBM Personal Computer AT is shown in Figure C-5.

Figure C-4 IBM PC and PC/XT U.S. Keyboard

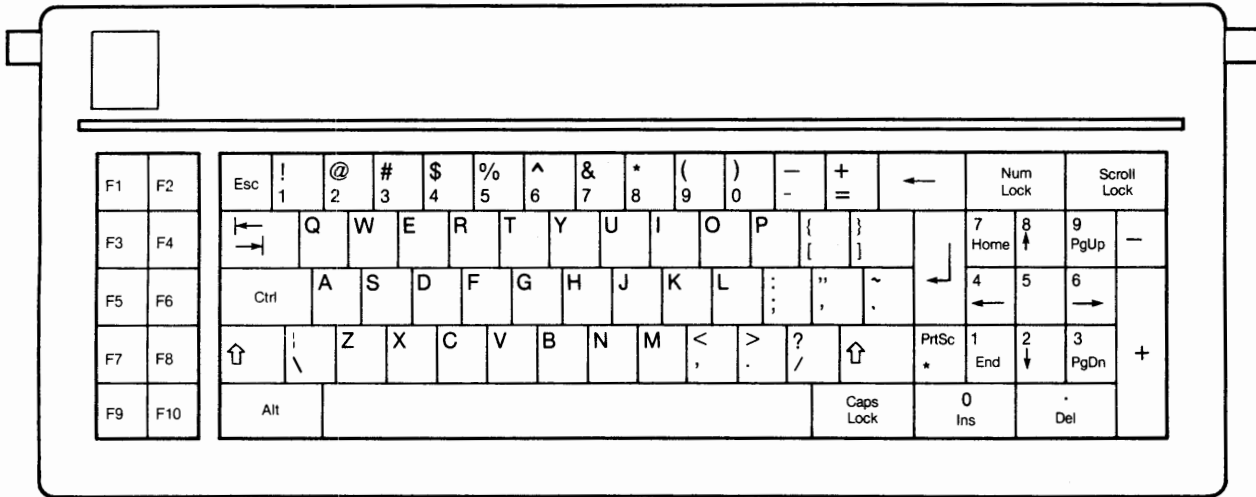


Figure C-5 IBM Personal Computer AT U.S. Keyboard

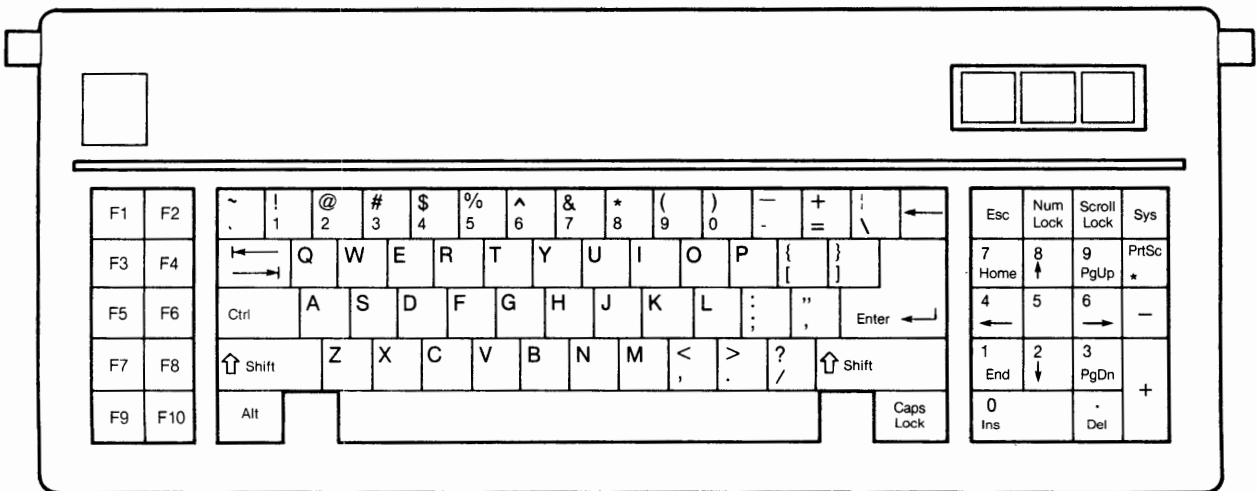


Table C-3 shows the key codes for the IBM Personal Computer. When referring to these tables, keep these notes in mind:

1. In each column, both the graphic and the hexadecimal value of the characters are given in the form ggg hh.
2. Entries consisting of — — indicate that the combination is suppressed within the keyboard DSR. These keys have no effect on Window Manager; that is, Window Manager does not return to the application program when any of these are pressed.
3. Entries consisting of xxx ** indicate special handling in the form of direct action by the keyboard DSR. There are three of these keys.
 - a. SHIFT-PRTSC — The print screen function DSR is performed unconditionally. This does not adversely affect Window Manager. Once the print screen function has finished, Window Manager resumes normal operation. Window Manager does not return to the application when the **SHIFT-PRTSC** keys are pressed.
 - b. CTRL-NUM LOCK — The DSR performs the pause function unconditionally. This does not adversely affect Window Manager. When another key is pressed, Window Manager resumes normal operation. Window Manager does not return to the application when the **CTRL-NUM LOCK** keys are pressed.
 - c. CTRL-BREAK — Window Manager returns to the application program. The key code returned is 100H. If the application then makes any call related to the operating system, the DSR program break function is performed.

The application can redirect the system interrupts to perform the application's own version of these three functions. Window Manager still handles the keys in the same manner, but the application's version of the function is performed instead of the normal system function.
4. Entries consisting of xxx yy* indicate that this key returns the extended code yy. If this type of key is pressed and it is unknown to Window Manager (not defined in Table C-2), Window Manager returns this extended code plus 100H.
5. All other entries are normal ASCII keys. If they are unknown to Window Manager, the usual code is returned.

Table C-3 Key Codes for the IBM Personal Computer

Scan Code	Normal		SHIFT		CTRL		ALT		Comments
01	ESC	1B	ESC	1B	ESC	1B	-	-	ESC
02	1	31	!	21	-	-	alt1	78*	
03	2	32	@	40	NUL	03*	alt2	79*	
04	3	33	#	23	-	-	alt3	7A*	
05	4	34	\$	24	-	-	alt4	7B*	
06	5	35	%	25	-	-	alt5	7C*	
07	6	36		5E	RS	1E	alt6	7D*	
08	7	37	&	26	-	-	alt7	7E*	
09	8	38	*	2A	-	-	alt8	7F*	
10	9	39	(28	-	-	alt9	80*	
11	0	30)	29	-	-	alt0	81*	
12	-	2D	_	5F	US	1F	alt-	82*	
13	=	3D	+	2B	-	-	alt=	83*	
14	BS	08	BS	08	DEL	7F	-	-	BACK SPACE
15	HT	09	Bktab	0F*	-	-	-	-	TAB
16	q	71	Q	51	DC1	11	altQ	10*	
17	w	77	W	57	ETB	17	altW	11*	
18	e	65	E	45	ENQ	05	altE	12*	
19	r	72	R	52	DC2	12	altR	13*	
20	t	74	T	54	DC4	14	altT	14*	
21	y	79	Y	59	EM	19	altY	15*	
22	u	75	U	55	NAK	15	altU	16*	
23	i	69	I	49	HT	09	altI	17*	
24	o	6F	O	4F	SI	0F	altO	18*	
25	p	70	P	50	DLE	10	altP	19*	
26	[5B	{	7B	ESC	1B	-	-	
27]	5D	}	7D	GS	1D	-	-	
28	CR	0D	CR	0D	LF	0A	-	-	ENTER
29	-	-	-	-	-	-	-	-	CTRL
30	a	61	A	41	SOH	01	altA	1E*	
31	s	73	S	53	DC3	13	altS	1F*	
32	d	64	D	44	EOT	04	altD	20*	
33	f	66	F	46	ACK	06	altF	21*	
34	g	67	G	47	BEL	07	altG	22*	
35	h	68	H	48	BS	08	altH	23*	
36	j	6A	J	4A	LF	0A	altJ	24*	
37	k	6B	K	4B	VT	0B	altK	25*	
38	l	6C	L	4C	FF	0C	altL	26*	
39	;	3B	:	3A	-	-	-	-	
40	'	27	"	22	-	-	-	-	
41	`	60	~	7E	-	-	-	-	
42	-	-	-	-	-	-	-	-	SHIFT
43	\	5C		7C	FS	1C	-	-	
44	z	7A	Z	5A	SUB	1A	altZ	2C*	
45	x	78	X	58	CAN	18	altX	2D*	
46	c	63	C	43	ETX	03	altC	2E*	

Table C-3 Key Codes for the IBM Personal Computer (Continued)

<i>Scan Code</i>	<i>Normal</i>		<i>SHIFT</i>		<i>CTRL</i>		<i>ALT</i>		<i>Comments</i>
47	v	76	V	56	SYN	16	altV	2F*	
48	b	62	B	42	STX	02	altB	30*	
49	n	6E	N	4E	SO	0E	altN	31*	
50	m	6D	M	4D	CR	0D	altM	32*	
51	,	2C	<	3C	—	—	—	—	
52	.	2E	>	3E	—	—	—	—	
53	/	2F	?	3F	—	—	—	—	
54	—	—	—	—	—	—	—	—	SHIFT
55	*	2A	***	**	Ptogl	72*	—	—	*/PRTSC
56	—	—	—	—	—	—	—	—	ALT
57	SP	20	SP	20	SP	20	SP	20	Space bar
58	—	—	—	—	—	—	—	—	CAPS LOCK
59	f1	3B*	sf1	54*	cf1	5E*	af1	68*	F1
60	f2	3C*	sf2	55*	cf2	5F*	af2	69*	F2
61	f3	3D*	sf3	56*	cf3	60*	af3	6A*	F3
62	f4	3E*	sf4	57*	cf4	61*	af4	6B*	F4
63	f5	3F*	sf5	58*	cf5	62*	af5	6C*	F5
64	f6	40*	sf6	59*	cf6	63*	af6	6D*	F6
65	f7	41*	sf7	5A*	cf7	64*	af7	6E*	F7
66	f8	42*	sf8	5B*	cf8	65*	af8	6F*	F8
67	f9	43*	sf9	5C*	cf9	66*	af9	70*	F9
68	f10	44*	sf10	5D*	cf10	67*	af10	71*	F10
69	—	—	—	—	Ppau	**	—	—	NUM LOCK/ PAUSE
70	—	—	—	—	Pbrk	**	—	—	SCRL LOCK/ BRK
71	Home	47*	7	37	cHome	77*	—	—	HOME/NUM 7
72	C-upp	48*	8	38	—	—	—	—	Up arrow/ NUM 8
73	Pg up	49*	9	39	cPgup	84*	—	—	Page up/ NUM 9
74	—	2D	—	2D	—	—	—	—	NUM —
75	C-lf	4B*	4	34	cC-lf	73*	—	—	Lf arrow/ NUM 4
76	—	—	5	35	—	—	—	—	NUM 5
77	C-rt	4D*	6	36	cC-rt	74*	—	—	Rt arrow/ NUM 6
78	+	2B	+	2B	—	—	—	—	NUM +
79	End	4F*	1	31	cEnd	75*	—	—	END/NUM 1
80	C-dn	50*	2	32	—	—	—	—	Dn Arrow/ NUM 2
81	Pg dn	51*	3	33	cPgdn	76*	—	—	Pg Down/ NUM 3
82	INS	52*	0	30	—	—	—	—	INS/NUM 0
83	DEL	53*	.	2E	—	—	—	—	DEL/NUM .

C INTERFACE

D

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
D.1	Introduction	D-3
D.2	Parameter Characteristics	D-3
D.2.1	Zero-Base Values	D-3
D.2.2	Integers	D-3
D.2.3	Character Strings	D-3
D.2.3.1	Passing Strings to Window Manager	D-4
D.2.3.2	Strings Returned by Window Manager	D-4
D.3	Callable Routines	D-5
D.3.1	Function Return Codes	D-5
D.4	Routines Called by Window Manager	D-14
D.5	C Compiling Requirements	D-17
D.5.1	Include File	D-17
D.6	Memory Considerations	D-19
D.7	Linking Considerations	D-20
D.7.1	Memory Model Differences	D-21
D.7.2	Dummy Routines Library	D-21
D.7.3	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files	D-21
D.7.4	Object Code Segments	D-22
D.8	Restrictions	D-22

Introduction

D.1 This appendix provides the information and procedures required to use the NaturalLink Window Manager with a C application. The C interface supports all models of the Lattice C Compiler. Generally, the parts of the NaturalLink package referenced from C procedures are linked with the C application program and act as an application program extension.

Parameter Characteristics

D.2 The only types of parameters used in the Window Manager call routines are integers and character strings. Integer parameters can be passed by either single-level reference (the address) or value. A string parameter is actually a copy of the pointer to the string since strings are never passed by value in C. The exact order of the parameters for each call is shown in Table D-1.

Zero-Base Values

D.2.1 For all C Window Manager calls that use a screen, window, or item identifier, the value of any of these identifiers is zero-base relative. In other words, the first valid screen, window, or item number is 0. The values correspond to the values assigned by Screen Builder.

Integers

D.2.2 Integers passed from C to the Window Manager routines can be declared as either integer or long integer. If long integers are passed, the upper 16 bits are ignored; it is recommended that long integers not be used.

Any integer parameters which are used for output are passed by reference (the address). These parameters are indicated by the ampersand (&) in front of the variable name in the procedure calls listed in Tables D-1 and D-2.

Character Strings

D.2.3 Window Manager strings are stored as eight-bit ASCII characters in sequential bytes of memory. A string pointer points to the first character, and the string is delimited by a null character (0 binary) or its maximum length.

When passing strings from C to the Window Manager interface, two arguments are required:

- The variable name of a string literal or character array
- The maximum length of the literal or array

Since C passes string arguments by address, the array name or string literal serves as the first argument. The following is an example of passing a character string.

```
STAT = WMSETS(SCR,WIND,ITEM, FIELD,"sample string",13);
```

The following example also illustrates the passing of a character string.

```
char SARRAY[80];  
SARRAY[0] = 't';  
SARRAY[1] = 'e';  
SARRAY[2] = 's';  
SARRAY[3] = 't';  
SARRAY[4] = '\0';  
STAT = WMSETS(SCR,WIND,ITEM, FIELD,SARRAY,80);
```

Passing Strings to Window Manager

D.2.3.1 The length parameter plays an important role when strings are passed to and from Window Manager. For reasons of speed, the length parameter always indicates the number of characters copied. Window Manager allocates the number of characters specified plus one character for a null terminator. After the copy is made, a check for the null terminating byte is made to find the real length of the string. If the string contains a null terminator, the null will indicate the end of the string; otherwise, the null appended to the string at length+1 by Window Manager will signify the string end.

In the first example in paragraph D.2.3, 13 characters are copied into the Window Manager data area. This is also the length of the string. In the second example, 80 characters are copied, but the null terminator indicates the string is actually only 4 characters long.

Strings Returned by Window Manager

D.2.3.2 The length parameter is of extreme importance when strings are passed back to the application program. Here again, Window Manager will copy back to the application the number of characters specified by the length parameter. The string will still be null terminated so a scan of the string will result in finding its true value. The length specified must take into account room for a null terminator.

This means that the buffer reserved by the application to hold the returned string must be as long or longer than the specified length parameter. If it is not, the application will risk having its data area written over. This means if you have an allocated buffer 10 characters long and specify a length of 20, even if the string to be returned is only 5 characters long, 20 characters will be returned and 10 bytes of memory will be trashed.

If the length parameter is not specified large enough for the string to be returned, an error will occur and Window Manager will pass back only the number of characters specified by the length parameter. The truncated string is still null terminated. This check is made only on the length parameter, not the buffer itself.

Callable Routines

D.3 Table D-1 correlates generic procedure calls to the corresponding C function calls. These are all of the calls that the application can make to Window Manager. For a summary of the calls that Window Manager makes to the application, see paragraph D.4, Routines Called by Window Manager. Table D-2 provides a specification of the C calling sequence, a description of the function, and the function parameters.

Function Return Codes

D.3.1 All of these function calls return a status code. The returned value from each C function is zero if no error was encountered or nonzero if an error condition occurred.

These nonzero codes are broken into two levels of severity. A negative code is a warning code. Warning codes are returned when the call did not complete as expected, but no harm was done. These codes can be ignored, but they indicate incorrect use of Window Manager and should be checked during the development process.

Positive codes indicate serious error conditions. When an application detects a positive code, the problem must be identified and corrected. Positive codes indicate that a parameter required to process the call was in error, processing was halted, and future calls will probably fail as well.

A message for any of the nonzero return codes can be displayed by calling the Display Message From Message Manager routine. The messages are stored in the LIERRMSG.NM\$ file. This filename must be supplied in the Message Manager call along with the code returned from the Window Manager call. LIERRMSG.NM\$ is included on the Window Manager object disk.

The Add Window, Select Window, Refresh Window, Display Window, Release Window, and Delete Window operations cannot return a detailed status code when operating on all of the windows in a screen. To be more specific, when the window number parameter of these calls is set to -1 , the status code returned indicates that an error or warning was encountered during operations on one or more windows. This status code implies a success or failure for the operations on all of the windows. When the window number parameter is set to a value equal to or greater than 0, a nonzero return status is a specific error condition.

The only operation performed on pop-up windows when the window number is set to -1 is the Delete operation. All other operations on pop-up windows must be performed with an individual call for each pop-up window.

Table D-1 C Interface Routines

Name: INITIALIZE WINDOW MANAGER
Call Sequence: WMINIT()
Description: Detects which machine the application is running on, initializes the video display attributes, installs keyboard mapping if needed, and loads the run-time version of the internal phrase file (NLXPHRAS.NM\$), if used. This routine **must** be called before any other Window Manager routines to ensure that Window Manager works correctly.
Parameters: None.

Name: RESET WINDOW MANAGER
Call Sequence: WMRSET()
Description: Restores the values set by the WMINIT procedure. It **must** be called to ensure the program terminates correctly.
Parameters: None.

CAUTION: WMRSET must be called prior to program termination. If it is not called, a system crash is likely to occur when the next program is executed. This is because WMRSET restores any keyboard mapping invoked by WMINIT and resets the cursor and video attributes. In addition to calling WMRSET before normal program termination, the application program should provide a way to call WMRSET in the event of a critical MS-DOS error in order to prevent a crash. For details on how to read the termination addresses, see the information on DOS Interrupts 22H and 23H in the *MS-DOS Operating System* manual.

Name: LOAD SCREEN FILE
Call Sequence: WMLOAD(&SCREEN, PATHNAME, LENGTH)
Description: Loads the screen description from the pathname into memory. A screen must be loaded before any of its windows can be manipulated by Window Manager.
Parameters:
 SCREEN: Output. Equal to -1 — Load failed. Integer greater than or equal to 0 — Integer assigned by Window Manager to identify screen.
 PATHNAME: Input. Character string that specifies the pathname of the screen description file.
 LENGTH: Input. Maximum length reserved for the pathname string.

Name: ADD WINDOW
Call Sequence: WMWADD(SCREEN, WINDOW)
Description: Makes a window known to Window Manager for subsequent operations.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Adds all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to add.

Table D-1 C Interface Routines (Continued)

Name: SELECT WINDOW
Call Sequence: **WMWSEL**(SCREEN, WINDOW)
Description: Makes a window active and flags the window for repainting, which will be performed with the next Receive call.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Select all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to select.

Name: REFRESH WINDOW
Call Sequence: **WMWREF**(SCREEN, WINDOW)
Description: Flags the window for text and border repainting. The window is redisplayed the next time a Receive call is performed.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify screen.
 WINDOW: Input. Integer equal to -1 — Refreshes all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to refresh.

Name: DISPLAY WINDOW
Call Sequence: **WMWDIS**(SCREEN, WINDOW)
Description: Repaints the window text and border without delay. A Receive call is not required for the Display procedure to execute.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Displays all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to display.

Table D-1 C Interface Routines (Continued)

Name: RECEIVE FROM WINDOW

Call Sequence: **WMWRCV**(&SCREEN, &WINDOW, &ITEM, &KEY, ITEXT, LENGTH)

Description: Receives user responses from an active window. Window Manager retains control until an item is selected or until an undefined key is pressed. At that time, Window Manager returns control to the application routine.

Parameters:

SCREEN: Input and output. The input integer is assigned by WMLOAD to identify the screen. The output integer specifies the screen of the window from which the response is received.

WINDOW: Input and output. The input integer specifies the window from which user response is desired. The output integer specifies the window of the item selected, or the window the cursor was in when an unknown key was pressed.

ITEM: Input and output. The input integer specifies the item upon which the cursor is initially placed. The output integer specifies the item selected.

KEY: Output. Code for the key pressed by the user.

ITEXT: Output. Character string to receive text of the selected item.

LENGTH: Input. Maximum length reserved for string (item text).

Name: RELEASE WINDOW

Call Sequence: **WMWREL**(SCREEN, WINDOW)

Description: Renders the window inactive and flags the window to have only text repainted. The window is redisplayed the next time a Receive call is performed.

Parameters:

SCREEN: Input. Integer assigned by WMLOAD to identify the screen.

WINDOW: Input. Integer equals -1 — Releases all active windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to release.

Name: DELETE WINDOW

Call Sequence: **WMWDEL**(SCREEN, WINDOW)

Description: Deletes the window from the screen and makes it unknown to Window Manager. If a window that covers other windows is deleted, the covered windows are flagged to have their borders and text repainted.

Parameters:

SCREEN: Input. Integer assigned by WMLOAD to identify the screen.

WINDOW: Input. Integer equals -1 — Deletes all windows for a screen, **including** pop-up windows. Integer greater than or equal to 0 — Window number to delete.

Table D-1 C Interface Routines (Continued)

Name: SAVE SCREEN
Call Sequence: **WMWSAV**(SCREEN, PATHNAME, LENGTH)
Description: Saves a screen to a file. This does not unload a screen from memory and any operation can be performed on any window in that screen after the SAVE procedure has been completed. This is useful if windows have been modified and they need to be saved.
Parameters:
 SCREEN: Input integer assigned by WMLOAD to identify screen file to be saved.
 PATHNAME: Input. String that specifies the pathname of the file where the screen is to be saved.
 LENGTH: Maximum length reserved for the pathname string.

Name: UNLOAD SCREEN
Call Sequence: **WMUNLD**(SCREEN)
Description: Releases the screen description from memory.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.

Name: ADD ITEM
Call Sequence: **WMIADD**(SCREEN, WINDOW, &ITEM)
Description: Adds an item to the end of the table and increases the size of the item table by 1.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Output. Item number of the added item.

Name: INSERT ITEM
Call Sequence: **WMIINS**(SCREEN, WINDOW, ITEM)
Description: Inserts a new item into the item table in front of the item position specified. All subsequent items are renumbered.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Input. Position in the item table where the item is to be inserted.

Table D-1 C Interface Routines (Continued)

Name:	DELETE ITEM
Call Sequence:	WMIDEL (SCREEN, WINDOW, ITEM)
Description:	Deletes an item from the item table. All subsequent items are renumbered.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
ITEM:	Input. Number of the item to be deleted.
<hr/>	
Name:	CREATE ITEM TABLE
Call Sequence:	WMICRE (SCREEN, WINDOW, NBR)
Description:	Creates an item table with the specified number of items. An item table can be created during initial window definition or during program execution. An item table is created when the current number of items is 0. It manipulates the item table based on the relationship between NBR and the current number of items in the window as follows: <ul style="list-style-type: none">■ Current number of items equals 0 — Creates an item table with NBR of items.■ NBR less than current number of items — Deletes items (current number items minus NBR) from the end of the item table. This condition returns a warning.■ NBR greater than current number of items — Adds items (NBR minus current number of items) to the end of the item table.■ NBR equal to 0 — Deletes the entire item table. This also returns a warning.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
NBR:	Input. Desired number of items to be created for the window.
<hr/>	
Name:	CREATE WINDOW
Call Sequence:	MMWCRE (&SCREEN, &WINDOW)
Description:	This routine creates a new window and adds it to the screen specified. If an invalid screen number is specified, a new screen will be created.
Parameters:	
SCREEN:	Input and output. Input integer assigned by WMLOAD to identify the screen to which the new window should be added. The output integer specifies the screen number of the new screen created if the input value was an invalid screen.
WINDOW:	Output. Window number of the window which was created.

Table D-1 C Interface Routines (Continued)

Name:	GET ATTRIBUTE VALUE
Call Sequence:	WMGETV (SCREEN, WINDOW, ITEM, FIELD, &IVALUE)
Description:	References the window structure and retrieves a value for the requested numeric attribute.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Item number of the item containing the field from which information is being requested. The value is ignored if the field is not associated with an item.
FIELD:	Input. Name of the attribute constant desired. See Table D-3 for a listing and description of the valid attribute constant names.
IVALUE:	Output. Current value of the requested attribute.
<hr/>	
Name:	GET STRING VALUE
Call Sequence:	WMGETS (SCREEN, WINDOW, ITEM, FIELD, STRING, LENGTH)
Description:	References the window structure and retrieves the string value for the requested string attribute.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Number of the item containing the field from which information is being requested. The value is ignored if the field is not associated with an item.
FIELD:	Input. This parameter uses one of the following attribute constant names: Lllabel Llshowf Llitlabl Llwmpath Llittext See Table D-3 for a description of the valid attribute constant names.
STRING:	Output. Character variable to receive returned string.
LENGTH:	Input. Maximum length reserved for the string.

Table D-1 C Interface Routines (Continued)

Name:	SET ATTRIBUTE VALUE
Call Sequence:	WMSETV(SCREEN, WINDOW, ITEM, FIELD, IVALUE)
Description:	References the window structure and sets the numeric attribute to the specified value.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Item number of the item having the attribute for which information is being specified. The value is ignored if the field is not associated with an item.
FIELD:	Input. Name of the attribute constant desired. See Table D-3 for a listing and description of the valid attribute constant names.
IVALUE:	Input. Integer value for the specified field.

CAUTION: No validation checks are performed on the IVALUE parameter in the WMSETV call. This reduces the code size of the high-level language interface. It is assumed that the value is in the correct range for the attribute being set. The other parameters—SCREEN, WINDOW, ITEM, and FIELD—are validated and error codes are returned if they are incorrect. See Section 3, Window Attributes, for the legal attribute values.

Name:	SET STRING VALUE
Call Sequence:	WMSETS(SCREEN, WINDOW, ITEM, FIELD, STRING, LENGTH)
Description:	References the window structure and sets a string attribute to the specified string.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Item number of the item having the attribute for which information is being specified. The value is ignored if the attribute is not associated with an item.
FIELD:	Input. This parameter uses one of the following attribute constant names: Llwlabel Llwshowf Llitolabl Llwmpath Llittext See Table D-3 for a description of the valid attribute constant names.
STRING:	Input. Character string constant containing the defined text.
LENGTH:	Input. Length of the string being entered.

Table D-1 C Interface Routines (Continued)

Name:	CLEAR SCREEN
Call Sequence:	WMCLRS()
Description:	Clears the text from the display area.
Parameters:	None.
<hr/>	
Name:	DISPLAY MESSAGE FROM MESSAGE MANAGER
Call Sequence:	DISMSG(MSGNUM, VARTXT, TXTLEN, VARSEP, MSGFIL, LENGTH)
Description:	Displays the message associated with a message number from a message file and inserts the variable text, if any is given, into the message.
Parameters:	
MSGNUM:	Input. Integer message number of the message that is to be displayed. The message number can be either a status from NaturalLink or a message built by the software designer using Message Builder.
VARTXT:	Input. String that contains a maximum of three variable text substrings to be inserted in the actual text message by Message Manager. If there is no variable text, the string is a null string. Each variable text substring must be separated by the character specified in the separator parameter. The separator character used cannot be a part of the variable text itself. For example, using the tilde as a separator would give: "1st variable text ~ 2nd variable text ~ 3rd variable text"
TXTLEN:	Input. Length of the variable text string.
VARSEP:	Input. Integer indicating the ASCII character which is used as the variable text separator. For example, a decimal value of 126 is passed to indicate the tilde ~ character.
MSGFIL:	Input. String that contains the message file pathname from which the text is to be retrieved.
LENGTH:	Input. Maximum length reserved for the message file pathname.
<hr/>	
Name:	RESET NATURALLINK MEMORY AREA
Call Sequence:	WMFLSH()
Description:	This routine clears out the memory area used by the NaturalLink run time. Any screens loaded or other data in this memory area will be lost and must be reloaded before NaturalLink processing can continue.
Parameters:	None.
<hr/>	

Routines Called by Window Manager

D.4 Table D-2 correlates generic procedure calls to the corresponding C function calls. These routines are supplied by the application and are called by Window Manager. If the functionality provided by these calls is not desired, the dummy routines provided on the Window Manager Runtime Object disk can be used instead. All of the application-provided calls except the validation call and the input call can return error status codes to Window Manager. If the status returned is nonzero, Window Manager will return to the application program with the status code.

Table D-2 Application-Provided Routines

Name:	APPLICATION VALIDATION ROUTINE
Call Sequence:	APPVAL(SCREEN, WINDOW, &ITEM, KEY, DATATYPE)
Description:	This routine is provided by the developer. Window Manager calls this routine when an edit field needs to be type-checked. It must return the status of the validation check. The defined statuses are as follows: -2 — Invalid/clear, type check failed, clear out edit field -1 — Invalid/no clear, type check failed, field untouched 0 — Valid, type check passed, continue 1 — Ignore, type check not done, return to edit field 2 — Exit, return to application immediately
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the window containing the item being validated.
ITEM:	Input and output. The input integer specifies the item to be validated. The output integer indicates the item the cursor should be placed on upon return to Window Manager.
KEY:	Input. Integer code for the key pressed by the user.
DATATYPE:	Input. Datatype assigned this item in the Set Item Attributes option in Screen Builder.
<hr/>	
Name:	APPLICATION DISPLAY WINDOW
Call Sequence:	APPDIS(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be displayed. This routine must display all aspects of the window except the window border.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to display.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table D-2 Application-Provided Routines (Continued)

Name:	APPLICATION RECEIVE WINDOW
Call Sequence:	APPRCV(SCREEN, WINDOW, &ITEM, &KEY, UDWNBR, &XPOS, &YPOS)
Description:	This routine is provided by the developer. Window Manager calls it when information must be received from a user-defined window. This routine must handle all user input for the given window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to receive information from.
ITEM:	Input and Output. The input integer specifies the item upon which the cursor should be initially placed. The output integer specifies the item selected.
KEY:	Output. Integer code for the key pressed by the user.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).
XPOS:	Output. Integer indicating the X character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.
YPOS:	Output. Integer indicating the Y character coordinate of the cursor. It is used only if the KEY parameter returned key code.

Name:	APPLICATION DELETE WINDOW
Call Sequence:	APPDEL(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be deleted. This routine must delete all nontext portions of the window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to delete.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table D-2 Application-Provided Routines (Continued)

Name:	APPLICATION DISPLAY MESSAGE
Call Sequence:	APPMSG(MSGSCR, MSGWND, &KEY, MSGNBR, MSGTYP, SCREEN, WINDOW, ITEM)
Description:	This routine is provided by the developer. It is called when a user-defined message needs to be displayed. This routine must handle the display of the message, response from the user to the message, and the deletion of the nontext portion of the message from the screen.
Parameters:	
MSGSCR:	Input. Integer assigned by Window Manager to identify the screen containing the message window.
MSGWND:	Input. Window number of the user-defined message window.
KEY:	Output. Integer code for the key pressed by the user.
MSGNBR:	Input. Number of the message to be displayed. This is the number assigned to the message by the Message Builder utility.
MSGTYP:	Input. Integer indicating the type of message being displayed. The integer is one of the following values: 5 — Help 6 — Error 7 — Warning 8 — Please Note
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen containing the window the cursor was in when the Help key was pressed. It is used only for Help messages.
WINDOW:	Input. Window number of the window the cursor was in when the Help key was pressed. It is used only for Help messages.
ITEM:	Input. Input integer specifies the item the cursor was on when the Help key was pressed. A value of -1 is passed when window-level help is being displayed. It is used only for Help messages.

Name:	APPLICATION INPUT ROUTINE
Call Sequence:	APPINP()
Description:	This routine is provided by the developer. Window Manager calls it when input from the user is needed. It can be used to provide an alternative input method to the keyboard. This routine, if used, must still return valid key codes. A value of 0 must be returned if there is no key code ready to be returned.
Parameters:	None.

C Compiling Requirements

D.5 Window Manager requires the use of an Include file that defines the attribute constants used by the application program. This is provided with the Window Manager package.

Include File D.5.1 A #define file containing attribute names for the WMSETV, WMGETV, WMSETS, and WMGETS Window Manager routines is provided to keep hard-coded constants out of the C code. These constants are likely to change or at least expand with future releases of Window Manager. The filename is WMFIELD.H, and the contents are given in Table D-3.

Table D-3 C Data Declarations for Window Attributes

```
/* Window coordinates: */

#define LIwposux 1 /* Left-most column */
#define LIwposuy 2 /* Top row */
#define LIwposlx 3 /* Right-most column */
#define LIwposly 4 /* Bottom row */

/* Window format: */

#define LIawdtyp 5 /* Window type */
#define LIwincol 38 /* Number of columns */
#define LIinwprio 10 /* Priority (for application use only) */
#define LIinwmusl 11 /* Multiple selection window */
#define LIinwpopu 12 /* Pop-up window */
#define LIinpant 46 /* Special repaint on receive */
#define LIwofset 21 /* First item to be displayed */
#define LIiautsc 40 /* Show last item when painted */
#define LIicentr 45 /* Center all items */
#define LIinorep 43 /* Don't redisplay current items */
#define LIinomul 42 /* Disable multiple line items */
#define LIicursin 75 /* Allow cursor to enter window */
#define LIidirct 44 /* Multiple column order */
#define LIilmlen 41 /* Maximum item label length */
#define LIiljust 47 /* Item label justification */
#define LIwshowf 20 /* Pathname of window file */
#define LIinwactv 13 /* Window is active (read only access) */
#define LIwitmb 39 /* Number of items (read only access) */

/* Active window attributes: */

#define LIablink 6 /* Blinking */
#define LIaunder 7 /* Underline */
#define LIarever 8 /* Reverse video */
#define LIactint 9 /* Intensity/color */

/* Inactive window attributes: */

#define LINwblnk 16 /* Blinking */
#define LINwundr 17 /* Underline */
#define LINwrevr 18 /* Reverse video */
#define LINwintn 19 /* Intensity/color */
```

Table D-3 C Data Declarations for Window Attributes (Continued)

```
/* Window Label attributes: */

#define LIwlabel 22 /* Window label */
#define LIwlvsbl 23 /* Invisible label */
#define LIwlpos 24 /* Label position */
#define LIwlcen 25 /* Centered label */
#define LIwblnk 27 /* Blinking label */
#define LIwlundr 28 /* Underlined label */
#define LIwlrevr 29 /* Reverse video label */
#define LIwlint 30 /* Intensity/color */
#define LIwlaint 26 /* Use item intensity */

/* Cursor attributes: */

#define LICrsiz 31 /* Cursor size */
#define LICblnk 33 /* Blinking cursor */
#define LICundr 34 /* Underlined cursor */
#define LICrever 35 /* Reverse video cursor */
#define LICurint 37 /* Cursor intensity/color */
#define LICalint 32 /* Use Item intensity */
#define LICstays 36 /* Cursor remains on a receive abort */

/* Border attributes: */

#define LINwbord 14 /* Bordered window */
#define LIBdrts 72 /* Border takes up space (0 = yes) */
#define LINwscmk 15 /* Display scroll markers */
#define LIBdrrv 73 /* Reverse video border */
#define LIBdrint 74 /* Border intensity/color */

/* Item attributes: */

#define LIittext 66 /* Item text */
#define LIitchos 56 /* Chosen/enable attributes */
#define LIitvsbl 63 /* Visible item */
#define LIitunsl 59 /* Unselectable item */
#define LIitdspl 55 /* Displayed */
#define LIitmlen 48 /* Maximum edit field length */
#define LIitreqr 57 /* Required edit field */
#define LIitecho 58 /* Echo edit field input */
#define LIidattyp 65 /* Edit field datatype */

/* Item chosen active attributes: */

#define LIitblnk 60 /* Blinking chosen item */
#define LIitundr 61 /* Underlined chosen item */
#define LIitrevr 62 /* Reverse video chosen item */
#define LIitint 64 /* Chosen item intensity/color */
```

Table D-3 C Data Declarations for Window Attributes (Continued)

```
/* Item label attributes: */

#define LIitlabl 54 /* Item label */
#define LIilblnk 50 /* Blinking label */
#define LIilundr 51 /* Underlined label */
#define LIilrevr 52 /* Reverse video label */
#define LIilint 53 /* Label intensity/color */
#define LIilaint 49 /* Use item intensity */

/* Other miscellaneous attributes: */

#define LIwmpath 71 /* Window Manager help/phrase file path */
#define LIpctype 76 /* Machine type (read only access) */
#define LIilcdon 77 /* PRO-LITE lcd screen indicator */
```

Memory Considerations

D.6 Since the NaturalLink Window Manager and the C application program coexist in memory, sufficient memory to accommodate the code and data areas of each is required.

To achieve correct ordering of the code and data areas for the C application program and Window Manager object, a set of routines has been added (one for each C model implemented for the application program) that defines the memory organization. Each of these routines places the application code in low memory, followed by the Window Manager code, the Window Manager data area, and finally the application's data area. Each routine also declares the Window Manager stack and heap size, which must be statically allocated. Since the application's stack and heap area occurs last, it can expand up to the physical memory limits of the machine. The routine names are LILCSSH (small program model), LILCPSH (large program model), LILCDSH (large data model), and LILCLSH (large model). Only one routine is selected to be included in the link stream.

Get-memory errors appear when the Window Manager heap management routines run out of memory. The only way to prevent these errors, if they occur, is to increase the size of the Window Manager heap area or to reduce the number of screens (or screen sizes) loaded into memory at the time the error occurred.

The default stack and heap sizes for Window Manager are as follows:

2K stack = hexadecimal 800 bytes
32K heap = hexadecimal 8000 bytes

The maximum size of the Window Manager stack and heap is the size in bytes determined by the difference of hexadecimal FFF0 and the size of the Window Manager data segment (called NL_DATA and found in the application's link map). It is recommended that at least hexadecimal 800 bytes be allocated for the Window Manager's stack size. Using this minimum recommendation for the stack size, the maximum amount of memory that can be allocated for the heap area is hexadecimal F7F0 minus the size of the data segment NL_DATA. This will be approximately hexadecimal ED00 bytes for application programs that use only the Window Manager object and slightly less for those that additionally use the Natural Language object. A change in the Window Manager stack and heap size will require changing the source file for one of the above-mentioned memory modules, reassembling the routine, and relinking the application program.

Linking Considerations

D.7 An example of a link control file used to link a simple C application program is as follows. Most of the NaturalLink runtime has been placed in libraries. The order of the libraries is significant. They must be placed in the order shown.

LILC?SH+	Memory organization module. This must occur first.
C?+	C program entry/exit module.
mymain+	The C application's main program.
...	Any application subroutines used.
demo /M	The .EXE file.
demo.MAP	The .MAP file.
mylib+	Any application libraries used.
LILC?+	Library of language interface routines to Window Manager for the Lattice C compiler.
WM+	Window Manager object library.
APP???+	Dummy Window Manager called application routines.
LC?	C run-time library.

**Memory
Model Differences**

D.7.1 Note that where a single question mark (?) appears in the module names, it is replaced by a letter designating which memory model is used by the application program, as follows:

- S for the small code/small data model
- P for the large code/small data model
- D for the small code/large data model
- L for the large code/large data model

The question marks in the APP??? library are also replaced by letters designating which memory model is used. If you are using the S model, APPLCS is used. APPLCD is used for the D model and APPOTH is used for both the P and the L model.

**Dummy
Routines Library**

D.7.2 The APP??? library contains dummy routines for all application routines which are called by Window Manager. These include APPVAL, APPRCV, APPDIS, APPDEL, APPMSG, and APPINP. If you do not use the features provided by one or more of these routines, the reference to them will be resolved by the dummy modules in this library. If you have your own versions of these routines they must be explicitly linked; if they are in a library, the library must come before APP??? in the link stream. If you create any libraries of your application routines, these libraries must come before any Naturallink libraries in the link stream.

**WMKEYDEF.OBJ
and
WMSTRDEF.OBJ
Files**

D.7.3 The key definition file (WMKEYDEF.OBJ) and the internal phrase file (WMSTRDEF.OBJ) are included as part of the Window Manager library (WM.LIB). If you have created your own version of these files (by using the KBUILD utility for WMKEYDEF or by reassembling the source file for WMSTRDEF), you must explicitly link your own version. Your version of the files should be placed in the link stream following any application subroutines that you may have. The linker will then use your versions instead of the files in the library.

Object Code Segments **D.7.4** The object code for Window Manager is divided into two groups. Segments with the name NL_PROG contain executable code, whereas segments with the name NL_DATA contain static data, heap, and the Window Manager run-time stack.

NOTE: Users who will be using PRE-2.00 versions of the Lattice C Compiler must use C.OBJ instead of C?.OBJ and LC.LIB instead of LC?.LIB in their link streams. Also, the memory module, language interface library, and the dummy library for small model Lattice C Compiler (that is, LILCSSH, LILCS, and APPLCS) should be used.

Restrictions **D.8** The maximum number of screens loaded from files is set to 20. A screen can have up to 255 windows and each window can have up to 65,536 items. Since screens are stored in the Window Manager heap, a screen which contains excessive windows and items can cause a get-memory error when loaded.

PASCAL INTERFACE

E

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
E.1	Introduction.....	E-3
E.2	Parameter Characteristics	E-3
E.2.1	Zero-Base Values	E-3
E.2.2	Integers	E-3
E.2.3	Character Strings	E-3
E.2.3.1	Passing Strings to Window Manager	E-4
E.2.3.2	Strings Returned by Window Manager	E-4
E.3	Callable Routines	E-5
E.3.1	Function Return Codes	E-5
E.4	Routines Called by Window Manager	E-14
E.5	Pascal Compiling Requirements	E-17
E.5.1	Include Files	E-17
E.6	Memory Considerations.....	E-19
E.7	Linking Considerations	E-20
E.7.1	Dummy Routines Library	E-20
E.7.2	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files.....	E-21
E.7.3	Object Code Segments	E-21
E.8	Restrictions.....	E-21

Introduction

E.1 This appendix provides the information and procedures required to use the NaturalLink Window Manager with an MS-Pascal application program. Generally, the parts of the NaturalLink package referenced from Pascal procedures are linked with the Pascal application program and act as an application program extension.

Parameter Characteristics

E.2 The only types of parameters used in the Window Manager call routines are integers and character strings. Integer parameters can either be passed by single level reference (the address) or by value. A string parameter is actually a copy of the pointer to the string, so strings will never be passed by value in Pascal calls. The exact order of the parameters is shown in Table E-1.

Zero-Base Values

E.2.1 For all Pascal Window Manager calls that use a screen, window, or item identifier, the value of any of these identifiers is zero-base relative. In other words, the first valid screen, window, or item number is 0. The values correspond to the values assigned by Screen Builder.

Integers

E.2.2 Integers passed from Pascal to the Window Manager routines can be declared as either integer or long integer. If long integers are passed, the upper 16 bits are ignored. It is recommended that these integers not be used.

Character Strings

E.2.3 Window Manager strings are stored as eight-bit ASCII characters in sequential bytes of memory. A string pointer points to the first character, and the string is delimited by a null character (0 binary) or its maximum length.

When passing strings from Pascal to the Window Manager interface, two arguments are required:

- The address of the first character in the string literal or array
- The value of the maximum length of the literal or array

Since Pascal must pass the address of the first character in the array, the array name subscripted with the index of the first character serves as the first argument. The following is an example of passing a character string.

```
SARRAY: PACKED ARRAY [1..80] OF CHAR;  
SARRAY[1] := 't';  
SARRAY[2] := 'e';  
SARRAY[3] := 's';  
SARRAY[4] := 't';  
SARRAY[5] := CHR(0);  
STAT := WMSETS(SCR,WIND,ITEM,FIELD,SARRAY[1],4);
```

The following example also illustrates the passing of a character string.

```
LENGTH := 80;  
STAT := WMSETS(SCR,WIND,ITEM,FIELD,SARRAY[1],LENGTH);
```

The Microsoft extensions to Pascal include two character array types, STRING and LSTRING. At present, the language interface does not support these types; hence, they cannot be used as parameters to the Window Manager Routines.

Passing Strings to Window Manager

E.2.3.1 The length parameter plays an important role when strings are passed to and from Window Manager. For reasons of speed, the length parameter always indicates the number of characters copied. Window Manager allocates the number of characters specified plus one character for a null terminator. After the copy is made, a check for the null terminating byte is made to find the real length of the string. If the string contained a null terminator, the null will indicate the end of the string; otherwise, the null appended to the string at length+1 by Window Manager will signify the string end.

In the first example in paragraph E.2.3, four characters are copied into Window Manager's data area. This is also the length of the string. In the second example, 80 characters are copied, but the null terminator indicates the string is actually only four characters long.

Strings Returned by Window Manager

E.2.3.2 The length parameter is of extreme importance when strings are passed back to the application program. Here again, Window Manager will copy back to the application the number of characters specified by the length parameter. The string will still be null terminated so a scan of the string will result in finding its true value. The length specified must take into account room for a null terminator.

This means that the buffer reserved by the application to hold the returned string must be as long or longer than the specified length parameter. If it is not, the application will risk having its data area written over. This means if you have an allocated buffer 10 characters long and specify a length of 20, even if the string to be returned is only 5 characters long, 20 characters will be returned and 10 bytes of memory will be trashed.

If the length parameter is not specified large enough for the string to be returned, an error will occur and Window Manager will only pass back the number of characters specified by the length parameter. The truncated string is still null terminated. This check is made only on the length parameter, not on the buffer itself.

Callable Routines

E.3 Table E-1 correlates generic procedure calls to the corresponding Pascal function calls. These are all the calls the application can make to Window Manager. For a summary of the calls that Window Manager makes to the application, see paragraph E.4, Routines Called by Window Manager. The table provides a specification of the Pascal calling sequence, a description of the function, and the function parameters. All routines are Pascal functions and must be declared and used as such.

Function Return Codes

E.3.1 All of the function calls return a status code. The returned value from each Pascal function is zero if no error was encountered or nonzero if an error condition occurred.

These nonzero codes are divided into two levels of severity. A negative code is a warning code. Warning codes are returned when the call did not complete as expected, but no harm was done. These codes can be ignored, but they indicate incorrect use of Window Manager and should be checked during the development process.

Positive return codes indicate serious error conditions. When an application detects a positive code, the problem must be identified and corrected. Positive codes indicate that something required to process the call was in error, processing was halted, and future calls will probably fail as well.

A message for any of the nonzero return codes can be displayed by calling the Display Message From Message Manager routine. The messages are stored in the LIERRMSG.NM\$ file. This is the filename that must be supplied in the Message Manager call along with the code returned from the Window Manager call. LIERRMSG.NM\$ is included on the Window Manager object disk.

The Add Window, Select Window, Refresh Window, Display Window, Release Window, and Delete Window operations cannot return a detailed status code when operating on all windows in a screen. To be more specific, when the window number parameter of these calls is set to -1 , the status code returned indicates that an error or warning was encountered during operations on one or more windows. This status code implies a success or failure for the operations on all the windows. When the window number parameter is set to a value equal to or greater than 0, a nonzero return status code indicates a specific error condition.

The only operation performed on pop-up windows when the window number is set to -1 is the Delete operation. All other operations on pop-up windows must be performed with an individual call for each pop-up window.

Table E-1 Pascal Interface Routines

Name: INITIALIZE WINDOW MANAGER
Call Sequence: **WMINIT**()
Description: Detects which machine the application is running on, initializes the video display attributes, installs keyboard mapping if needed, and loads the run-time version of the internal phrase file (NLXPHRAS.NM\$), if used. This routine **must** be called before any other Window Manager routines to ensure that Window Manager works correctly.
Parameters: None

Name: WMRSET WINDOW MANAGER
Call Sequence: **WMRSET**()
Description: Restores the values set by the WMINIT procedure. It **must** be called to ensure that the program terminates correctly.
Parameters: None

CAUTION: WMRSET must be called prior to program termination. If it is not called, a system crash is likely to occur when the next program is executed. This is because WMRSET restores any keyboard mapping invoked by WMINIT and resets the cursor and video attributes. In addition to calling WMRSET before normal program termination, the application program should provide a way to call WMRSET in the event of a critical MS-DOS error in order to prevent a crash. For details on how to trap the termination addresses, see the information on DOS Interrupts 22H and 23H in the *MS-DOS Operating System* manual.

Name: LOAD SCREEN FILE
Call Sequence: **WMLOAD**(SCREEN, PATHNAME[1], LENGTH)
Description: Loads the screen description from the pathname into memory. A screen must be loaded before any of its windows can be manipulated by Window Manager.
Parameters:
 SCREEN: Output. Equal to -1 — Load failed. Integer greater than or equal to 0 — Integer assigned by Window Manager to identify screen.
 PATHNAME: Input. Character string that specifies the pathname of the screen description file.
 LENGTH: Input. Maximum length reserved for the pathname string.

Name: ADD WINDOW
Call Sequence: **WMWADD**(SCREEN, WINDOW)
Description: Makes a window known to Window Manager for subsequent operations.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Adds all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to add.

Table E-1 Pascal Interface Routines (Continued)

Name: SELECT WINDOW
Call Sequence: **WMWSEL**(SCREEN, WINDOW)
Description: Makes a window active and flags the window for repainting, which will be performed with the next Receive call.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Select all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to select.

Name: REFRESH WINDOW
Call Sequence: **WMWREF**(SCREEN, WINDOW)
Description: Flags the window for text and border repainting. The window is redisplayed the next time a Receive call is performed.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify screen.
 WINDOW: Input. Integer equal to -1 — Refreshes all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to refresh.

Name: DISPLAY WINDOW
Call Sequence: **MMWDIS**(SCREEN, WINDOW)
Description: Repaints the window text and border without delay. A Receive call is not required for the Display procedure to execute.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Displays all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to display.

Name: RECEIVE FROM WINDOW
Call Sequence: **MMWRCV**(SCREEN, WINDOW, ITEM, KEY, ITEXT[1], LENGTH)
Description: Receives user responses from an active window. Window Manager retains control until an item is selected or until an undefined key is pressed. At that time, Window Manager returns control to the application routine.

Table E-1 Pascal Interface Routines (Continued)

Parameters:

SCREEN:	Input and output. The input integer is assigned by WMLOAD to identify the screen. The output integer specifies the screen of the window from which the response is received.
WINDOW:	Input and output. The input integer specifies from which window user response is desired. The output integer specifies the window of the item selected, or the window the cursor was in when an unknown key was pressed.
ITEM:	Input and output. The input integer specifies the item upon which the cursor is initially placed. The output integer specifies the item selected.
KEY:	Output. Code for the key pressed by the user.
ITEXT:	Output. Character string to receive text of the selected item.
LENGTH:	Input. Maximum length reserved for string (item text).

Name: RELEASE WINDOW

Call Sequence: **WMWREL**(SCREEN, WINDOW)

Description: Renders the window inactive and flags the window to have only text repainted. The window is redisplayed the next time a Receive call is performed.

Parameters:

SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Integer equals -1 — Releases all active windows for a screen, except pop-up windows. Integer greater than or equal to 0 — Window number to release.

Name: DELETE WINDOW

Call Sequence: **MMWDEL**(SCREEN, WINDOW)

Description: Deletes the window from the screen and makes it unknown to Window Manager. If a window that covers other windows is deleted, the covered windows are flagged to have their borders and text repainted.

Parameters:

SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Integer equals -1 — Deletes all windows for a screen, including pop-up windows. Integer greater than or equal to 0 — Window number to delete.

Name: SAVE SCREEN

Call Sequence: **MMWSAV**(SCREEN, PATHNAME[1], LENGTH)

Description: Saves a screen to a file. This does not unload a screen from memory; any operation can be performed on any window in that screen after the SAVE procedure has been completed. This is useful if windows have been modified and they need to be saved.

Parameters:

SCREEN:	Input integer assigned by WMLOAD to identify the screen file to be saved.
PATHNAME:	Input. String that specifies the pathname of the file where the screen is to be saved.
LENGTH:	Maximum length reserved for the pathname string.

Table E-1 Pascal Interface Routines (Continued)

Name: UNLOAD SCREEN
Call Sequence: **WMUNLD**(SCREEN)
Description: Releases the screen description from memory.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.

Name: ADD ITEM
Call Sequence: **WMIADD**(SCREEN, WINDOW, ITEM)
Description: Adds an item to the end of the table and increases the size of the item table by 1.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Output. Item number of the added item.

Name: INSERT ITEM
Call Sequence: **WMIINS**(SCREEN, WINDOW, ITEM)
Description: Inserts a new item into the item table in front of the item position specified. All subsequent items are renumbered.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Input. Position in the item table where the item is to be inserted.

Name: DELETE ITEM
Call Sequence: **WMIDEL**(SCREEN, WINDOW, ITEM)
Description: Deletes an item from the item table. All subsequent items are renumbered.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Input. Number of the item to be deleted.

Table E-1 Pascal Interface Routines (Continued)

Name:	CREATE ITEM TABLE
Call Sequence:	WMICRE (SCREEN, WINDOW, NBR)
Description:	Creates an item table with the specified number of items. An item table can be created during initial window definition or during program execution. An item table is created when the current number of items is 0. It manipulates the item table based on the relationship between NBR and the current number of items in the window as follows: <ul style="list-style-type: none">■ Current number of items equals 0 — Creates an item table with NBR of items.■ NBR less than current number of items — Deletes items (current number items minus NBR) from the end of the item table. This condition returns a warning.■ NBR greater than current number of items — Adds items (NBR minus current number of items) to the end of the item table.■ NBR equal to 0 — Deletes the entire item table. This also returns a warning.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
NBR:	Input. Desired number of items to be created for the window.
<hr/>	
Name:	CREATE WINDOW
Call Sequence:	WMWCRE (SCREEN, WINDOW)
Description:	This routine creates a new window and adds it to the screen specified. If an invalid screen number is specified, a new screen will be created.
Parameters:	
SCREEN:	Input and output. This is the input integer assigned by WMLOAD to identify the screen to which the new window should be added. The output integer specifies the screen number of the new screen created if the input value was an invalid screen.
WINDOW:	Output. Window number of the window which was created.
<hr/>	
Name:	GET ATTRIBUTE VALUE
Call Sequence:	WMGETV (SCREEN, WINDOW, ITEM, FIELD, IVALUE)
Description:	References the window structure and retrieves a value for the requested numeric attribute.

Table E-1 Pascal Interface Routines (Continued)

Parameters:

SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Number of the item containing the field from which information is being requested. The value is ignored if the field is not associated with an item.
FIELD:	Input. Name of the attribute constant desired. See Table E-3 for a listing and description of the valid attribute constant names.
IVALUE:	Output. Current value of the requested attribute.

Name: GET STRING VALUE

Call Sequence: WMGETS(SCREEN, WINDOW, ITEM, FIELD, STRING[1], LENGTH)

Description: References the window structure and retrieves the string value for the requested string attribute.

Parameters:

SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Item number of the item containing the field from which information is being requested. The value is ignored if the field is not associated with an item.
FIELD:	Input. This parameter uses one of the following attribute constant names: Llwstring Llwstringf Llwstringl Llwstringpath Llwstringtext See Table E-3 for a description of the valid attribute constant names.
STRING:	Output. Character variable to receive returned string.
LENGTH:	Input. Maximum length reserved for the string.

Name: SET ATTRIBUTE VALUE

Call Sequence: WMSETV(SCREEN, WINDOW, ITEM, FIELD, IVALUE)

Description: References the window structure and sets the numeric attribute to the specified value.

Table E-1 Pascal Interface Routines (Continued)

Parameters:

SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Item number of the item having the attribute for which information is being specified. The value is ignored if the field is not associated with an item.
FIELD:	Input. Name of the attribute constant desired. See Table E-3 for a listing and description of the valid attribute constant names.
IVALUE:	Input. Integer value for the specified field.

CAUTION: No validation checks are performed on the IVALUE parameter in the WMSETV call. This reduces the code size of the high-level language interface. The value is assumed to be in the correct range for the attribute being set. The other parameters — Screen, Window, Item, and Field — are validated and error codes are returned if they are incorrect. See Chapter 3, Window Attributes, for the legal attribute values.

Name: SET STRING VALUE

Call Sequence: WMSETS(SCREEN, WINDOW, ITEM, FIELD, STRING[1], LENGTH)

Description: References the window structure and sets a string attribute to the specified string.

Parameters:

SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Item number of the item having the attribute for which information is being specified. The value is ignored if the attribute is not associated with an item.
FIELD:	Input. This parameter uses one of the following attribute constant names: LIwlabel LIwshowf LIltabl LIwmpath LIltext See Table E-3 for a description of the valid attribute constant names.
STRING:	Input. Character string constant containing the defined text.
LENGTH:	Input. Length of the string being entered.

Table E-1 Pascal Interface Routines (Continued)

Name:	CLEAR SCREEN
Call Sequence:	WMCLRS()
Description:	Clears the text from the display area.
Parameters:	None
<hr/>	
Name:	DISPLAY MESSAGE FROM MESSAGE MANAGER
Call Sequence:	DISMSG(MSGNUM, VARTX[1], TXTLEN, VARSEP, MSGFIL[1], LENGTH)
Description:	Displays the message associated with a message number from a message file and inserts the variable text, if any is given, into the message.
Parameters:	
MSGNUM:	Input. Integer message number of the message that is to be displayed. The message number can be either a status from NaturalLink or a message built by the software designer using Message Builder.
VARTXT:	Input. String that contains a maximum of three variable text substrings to be inserted in the actual text message by Message Manager. If there is no variable text, the string is a null string. Each variable text substring must be separated by the character specified in the separator parameter. The separator character used cannot be a part of the variable text itself. For example, using the tilde as a separator would give: "1st variable text ~ 2nd variable text ~ 3rd variable text".
TXTLEN:	Input. Length of the variable text string.
VARSEP:	Input. Integer indicating the ASCII character which is used as the variable text separator. For example, a decimal value of 126 is passed to indicate the tilde ~ character.
MSGFIL:	Input. String that contains the message file pathname from which the text is to be retrieved.
LENGTH:	Input. Maximum length reserved for the message file pathname.
<hr/>	
Name:	RESET NATURALLINK MEMORY AREA
Call Sequence:	WMFLSH()
Description:	This routine clears out the memory area used by the NaturalLink run time. Any screens loaded or any other data in this memory area will be lost and must be reloaded before NaturalLink processing can continue.
Parameters:	None.

Routines Called by Window Manager

E.4 Table E-2 correlates generic procedure calls to the corresponding C function calls. These routines are supplied by the application and are called by Window Manager. If the functionality provided by these calls is not desired, the dummy routines provided on the Window Manager Run-Time Object disk can be used instead. All of the application-provided calls except the validation call and the input call can return error status codes to Window Manager. If the status returned is nonzero, Window Manager will return to the application program with the status code.

Table E-2 Application-Provided Routines

Name:	APPLICATION VALIDATION ROUTINE
Call Sequence:	APPVAL(SCREEN, WINDOW, ITEM, KEY, DATATYPE)
Description:	This routine is provided by the developer. Window Manager calls this routine when an edit field needs to be type-checked. It must return the status of the validation check. The defined statuses are as follows: -2 — Invalid/clear, type check failed, clear out edit field -1 — Invalid/no clear, type check failed, field untouched 0 — Valid, type check passed, continue 1 — Ignore, type check not done, return to edit field 2 — Exit, return to application immediately
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the window containing the item being validated.
ITEM:	Input and output. The input integer specifies the item to be validated. The output integer indicates which item the cursor should be placed on upon return to Window Manager.
KEY:	Input. Integer code for the key pressed by the user.
DATATYPE:	Input. Datatype assigned this item in the Set Item Attributes option in Screen Builder.
<hr/>	
Name:	APPLICATION DISPLAY WINDOW
Call Sequence:	APPDIS(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be displayed. This routine must display all aspects of the window except the window border.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to display.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table E-2 Application-Provided Routines (Continued)

Name:	APPLICATION RECEIVE WINDOW
Call Sequence:	APPRCV(SCREEN, WINDOW, ITEM, KEY, UDWNBR, XPOS, YPOS)
Description:	This routine is provided by the developer. Window Manager calls it when information must be received from a user-defined window. This routine must handle all user input for the given window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window from which to receive information.
ITEM:	Input and Output. The input integer specifies the item upon which the cursor should be initially placed. The output integer specifies the item selected.
KEY:	Output. Integer code for the key pressed by the user.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).
XPOS:	Output. This is the integer indicating the X character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.
YPOS:	Output. This is the integer indicating the Y character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.
<hr/>	
Name:	APPLICATION DELETE WINDOW
Call Sequence:	APPDEL(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be deleted. This routine must delete all nontext portions of the window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to be deleted.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).
<hr/>	
Name:	APPLICATION DISPLAY MESSAGE
Call Sequence:	APPMSG(MSGSCR, MSGWND, KEY, MSGNBR, MSGTYP, SCREEN, WINDOW, ITEM)
Description:	This routine is provided by the developer. It is called when a user-defined message needs to be displayed. This routine must handle the display of the message, response from the user to the message, and the deletion of the nontext portion of the message from the screen.

Table E-2 Application-Provided Routines (Continued)

Parameters:

MSGSCR:	Input. Integer assigned by Window Manager to identify the screen containing the message window.
MSGWND:	Input. Window number of the user-defined message window.
KEY:	Output. Integer code for the key pressed by the user.
MSGNBR:	Input. Number of the message to be displayed. This is the number assigned to the message by the Message Builder utility.
MSGTYP:	Input. Integer indicating the type of message being displayed. The integer is one of the following values: 5 — Help 6 — Error 7 — Warning 8 — Please Note
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen containing the window the cursor was in when the HELP key was pressed. It is used only for Help messages.
WINDOW:	Input. Window number of the window the cursor was in when the HELP key was pressed. It is used only for Help messages.
ITEM:	Input. Input integer specifies the item the cursor was on when the HELP key was pressed. A value of -1 is passed when window level help is being displayed. It is used only for Help messages.

Name: APPLICATION INPUT ROUTINE

Call Sequence: APPINP()

Description: This routine is provided by the developer. Window Manager calls it when input from the user is needed. It can be used to provide an alternative input method to the keyboard. This routine, if used, must still return valid key codes. A value of 0 must be returned if there is no key code ready to be returned.

Parameters: None.

Pascal Compiling Requirements

E.5 Window Manager requires the use of Include files which define the attribute constants used by the application program and the Window Manager routine declarations. This is provided with the Window Manager package.

Include Files E.5.1 The declarations for each of the Window Manager routines can be found in the WMEXTRNS.PAS file and must be included in the Pascal application.

A constant declaration file containing attribute names for the WMSETV, WMGETV, WMSETS, and WMGETS Window Manager routines is provided to keep hard-coded constants out of the Pascal code. These constants are likely to change or at least expand with future releases of Window Manager. The filename is WMFIELD.CON, and the contents are given in Table E-3.

Table E-3 Pascal Data Declarations for Window Attributes

CONST

{* Window Coordinates: *}

```
LIwposux = 1;  {*} Left-most column          *}
LIwposuy = 2;  {*} Top row                    *}
LIwposlx = 3;  {*} Right-most column         *}
LIwposly = 4;  {*} Bottom row                *
```

{* Window format: *}

```
LIawdtyp = 5;  {*} Window type                *}
LIwincol = 38;  {*} Number of columns          *}
LIinwprio = 10;  {*} Priority (for application use only) *}
LIinwmul = 11;  {*} Multiple selection window   *}
LIinwpopu = 12;  {*} Pop-up window             *}
LIinpant = 46;  {*} Special repaint on receive  *}
LIwofset = 21;  {*} First item to be displayed  *}
LIiautsc = 40;  {*} Show last item when painted *}
LIicentr = 45;  {*} Center all items           *}
LIinorep = 43;  {*} Don't redisplay current items *}
LIinomul = 42;  {*} Disable multiple line items *}
LIcursin = 75;  {*} Allow cursor to move into window *}
LIidirct = 44;  {*} Multiple column order      *}
LIilmlen = 41;  {*} Maximum item label length  *}
LIiljust = 47;  {*} Item label justification   *}
LIwshowf = 20;  {*} Pathname of window file    *}
LIwactv = 13;  {*} Window is active (read only access) *}
LIwitmb = 39;  {*} Number of items (read only access) *
```

Table E-3 Pascal Data Declarations for Window Attributes (Continued)

```
{* Active window attributes: *}

LIablink = 6;  {*} Blinking                *}
LIaunder = 7;  {*} Underlined              *}
LIarever = 8;  {*} Reverse video          *}
LIactint = 9;  {*} Intensity/color        *}

{* Inactive window attributes: *}

LINwblnk = 16;  {*} Blinking                *}
LINwundr = 17;  {*} Underlined              *}
LINwrevr = 18;  {*} Reverse video          *}
LINwintn = 19;  {*} Intensity/color        *}

{* Window Label attributes: *}

LIwlabel = 22;  {*} Window label           *}
LIwlvsbl = 23;  {*} Invisible label        *}
LIwlpos  = 24;  {*} Label position         *}
LIwlcent = 25;  {*} Centered label        *}
LIwlblnk = 27;  {*} Blinking label        *}
LIwlundr = 28;  {*} Underlined label      *}
LIwlrevr = 29;  {*} Reverse video label   *}
LIwlint  = 30;  {*} Intensity/color       *}
LIwlaint = 26;  {*} Use item intensity    *}

{* Cursor attributes: *}

LIcrsize = 31;  {*} Cursor size            *}
LIcblink = 33;  {*} Blinking cursor       *}
LIcunder = 34;  {*} Underlined cursor     *}
LIcrever = 35;  {*} Reverse video cursor  *}
LIcurint  = 37;  {*} Cursor intensity/color *}
LIcalint  = 32;  {*} Use Item intensity    *}
LIcstays  = 36;  {*} Cursor remains on a receive abort *}

{* Border attributes: *}

LINwbord = 41  {*} Bordered window        *}
LIbdrtsp = 72  {*} Border takes up space (0 = Yes) *}
LIwscmk  = 15  {*} Display scroll markers   *}
LIbdrrv  = 73  {*} Reverse video border   *}
LIbdrint = 74  {*} Border intensity/color  *}

{* Item attributes: *}

LIittext = 66;  {*} Item text              *}
LIitchos = 56;  {*} Chosen/enable attributes *}
LIitvsbl = 63;  {*} Visible item          *}
LIitunsl = 59;  {*} Unselectable item     *}
LIitdspl = 55;  {*} Displayed             *}
LIitmten = 48;  {*} Maximum edit field length *}
LIitreqr = 57;  {*} Required edit field   *}
LIitecho = 58;  {*} Echo edit field input  *}
LIidattyp = 65;  {*} Edit field datatype   *}
```

Table E-3 Pascal Data Declarations for Window Attributes (Continued)

```
{* Item chosen active attributes: *}

LIitblnk = 60;  {*} Blinking chosen item          *}
LIitundr = 61;  {*} Underlined chosen item        *}
LIitrevr = 62;  {*} Reverse video chosen item    *}
LIitint  = 64;  {*} Chosen item intensity/color   *}

{* Item label attributes: *}

LIitlabl = 54;  {*} Item label                    *}
LIilblnk = 50;  {*} Blinking label                *}
LIilundr = 51;  {*} Underlined label              *}
LIilrevr = 52;  {*} Reverse video label          *}
LIilint  = 53;  {*} Label intensity/color        *}
LIilaint = 49;  {*} Use item intensity           *}

{* Other miscellaneous attributes: *}

LIwmpath = 71;  {*} Window Manager help/phrase file path  *}
LIpctype = 76;  {*} Machine type (read only access)      *}
LIlcdon  = 77;  {*} PRO-LITE lcd screen indicator        *}
```

Memory Considerations

E.6 Since the NaturalLink Window Manager and the Pascal application program coexist in memory, sufficient memory to accommodate the code and data areas of each is required.

To achieve correct ordering of the code and data areas for the Pascal application program and Window Manager object, a routine called LIMSPSH has been added that defines the memory organization. This routine places the application code in low memory, followed by the Window Manager code, the Window Manager data area, and finally the application's data area. This routine declares the Window Manager stack and heap size, which must be statically allocated. Since the application program's stack and heap area occurs last, it can expand up to the physical memory limits of the machine.

Get-memory errors appear when the Window Manager heap management routines run out of memory. The only way to prevent these errors, if they occur, is to increase the size of the Window Manager heap area or to reduce the number of screens (or screen sizes) loaded into memory at the time the error occurred.

The default stack and heap sizes for Window Manager are as follows:

2K stack = hexadecimal 800 bytes
32K heap = hexadecimal 8000 bytes

The maximum size of the Window Manager stack and heap is the size in bytes determined by the difference of hexadecimal FFF0 and the size of the Window Manager data segment (called NL_DATA and found in the application's link map). At least hexadecimal 800 bytes should be allocated for the Window Manager stack size. Using this minimum recommendation for the stack size, the maximum amount of memory that can be allocated for the heap area is hexadecimal F7F0 minus the size of the data segment NL_DATA. This will be approximately hexadecimal ED00 bytes for applications that use only the Window Manager object and slightly less for those that additionally use Natural Language object. A change in the Window Manager stack and heap size will require changing the LIMSPSH source file, reassembling the routine, and relinking the application program.

Linking Considerations

E.7 An example of an MS-DOS link control file used to link a simple Pascal application program follows. Most of the NaturalLink runtime has been placed in libraries. The order of the libraries is significant, and they must be placed in the order shown.

LIMSPSH+	Memory organization module. This must occur first.
mymain+	The Pascal application's main program.
...	Any application subroutines, if used.
demo /M	The .EXE file.
demo.MAP	The .MAP file.
mylib+	Any application libraries, if used.
LIMSP+	Library of language interface routines to Window Manager for Pascal.
WM+	Window Manager object library.
APPOTH+	Dummy Window Manager called application routines.
PASCAL	Pascal run-time library.

Dummy Routines Library

E.7.1 The APPOTH library contains dummy routines for all application routines which are called by Window Manager. These include APPVAL, APPRCV, APPDIS, APPDEL, APPMSG, and APPINP. If you do not use the features provided by one or more of these routines, the reference to them will be resolved by the dummy modules in this library. If you have your own versions of these routines, they must be explicitly linked; if they are in a library, the library must come before APPOTH in the link stream. If you create any libraries of your application routines, these libraries must come before any NaturalLink libraries in the link stream.

**WMKEYDEF.OBJ
and
WMSTRDEF.OBJ
Files**

E.7.2 The key definition file (WMKEYDEF.OBJ) and the internal phrase file (WMSTRDEF.OBJ) are included as part of the Window Manager library (WM.LIB). If you have created your own version of these files (by using the KBUILD utility for WMKEYDEF or by reassembling the source file for WMSTRDEF), you must explicitly link your own version. Your version of the files should be placed in the link stream following any application subroutines that you may have. The linker will then use your versions instead of the files in the library.

**Object Code
Segments**

E.7.3 The object code for Window Manager is divided into two groups. Segments with the name NL_PROG contain executable code, while segments with the name NL_DATA contain static data, heap, and Window Manager run-time stack.

Restrictions

E.8 The maximum number of screens loaded from files is set to 20. A screen can have up to 255 windows and each window can have up to 65,536 items. Since screens are stored in the Window Manager heap, a screen that contains excessive windows and items can cause a get-memory error when loaded.

FORTRAN INTERFACE

F

<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
F.1	Introduction.....	F-3
F.2	Parameter Characteristics	F-3
F.2.1	One-Base Values	F-3
F.2.2	Integers.....	F-3
F.2.3	Character Strings	F-3
F.2.3.1	Passing Strings to Window Manager	F-4
F.2.3.2	Strings Returned by Window Manager	F-4
F.3	Callable Routines.....	F-5
F.3.1	Function Return Codes	F-5
F.4	Routines Called by Window Manager	F-14
F.5	FORTRAN Compiling Requirements	F-17
F.5.1	Common Blocks and Include Files	F-17
F.5.2	Initialization of Window Manager	F-17
F.6	Memory Considerations.....	F-20
F.7	Linking Considerations	F-21
F.7.1	Dummy Routines Library	F-21
F.7.2	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files.....	F-21
F.7.3	Object Code Segments.....	F-21
F.8	Restrictions.....	F-22

Introduction

F.1 This appendix presents the information and procedures required to use the NaturalLink Window Manager with an MS-FORTRAN application program. Generally, the parts of the NaturalLink package referenced from FORTRAN procedures are linked with the FORTRAN application program and act as an application program extension.

Parameter Characteristics

F.2 The only types of parameters used in the Window Manager call routines are integers and character strings. The exact order of these parameters is shown in Table F-1, FORTRAN Interface Routines.

One-Base Values

F.2.1 For all FORTRAN Window Manager calls that use a screen, window, or item identifier, the value of these identifiers is one-base relative. In other words, a screen, window, or item number that is 0 is illegal, and results in an error being returned by the interface. This one-base indexing is different than the zero-base indexing used by Screen Builder and other high-level languages. The smallest possible screen, window, or item number is 1. This corresponds to window 0 and item 0 as assigned by Screen Builder.

Integers

F.2.2 Integers passed from FORTRAN to the Window Manager interface can be either INTEGER*2 or INTEGER*4. If INTEGER*4 integers are passed, the upper 16 bits are ignored. It is recommended that these integers not be used.

Character Strings

F.2.3 Window Manager strings are stored as eight-bit ASCII characters in sequential bytes of memory. A string pointer points to the first character, and the string is delimited by a null character (0 binary).

When passing strings from FORTRAN to the Window Manager interface, two arguments are required:

- The variable name of a string literal or character array
- The maximum length of the literal or array

Since FORTRAN passes function arguments by address, the array name or string literal serves as the first argument. The following example illustrates the passing of a character string.

```
STAT = WMSETS(SCR,WIND,ITEM,FIELD,'sample string',13)
```

The following example also illustrates the passing of a character string.

```
CHARACTER*1 SARRAY(80)
SARRAY(1) = 't'
SARRAY(2) = 'e'
SARRAY(3) = 's'
SARRAY(4) = 't'
SARRAY(5) = 0
STAT = WMSETS(SCR,WIND,ITEM,FIELD,SARRAY,80)
```

Passing Strings to Window Manager

F.2.3.1 The length parameter plays an important role when strings are passed to and from Window Manager. For reasons of speed, the length parameter always indicates the number of characters copied. Window Manager allocates the number of characters specified plus one character for a null terminator. After the copy is made, a check for the null terminating byte is made to find the real length of the string. If the string contained a null terminator, the null will indicate the end of the string; otherwise, the null appended to the string at length+1 by Window Manager will signify the string end.

In the first example in paragraph F.2.3, 13 characters are copied into the Window Manager data area. This is also the length of the string. In the second example, 80 characters are copied, but the null terminator indicates the string is actually only four characters long.

Strings Returned by Window Manager

F.2.3.2 The length parameter is of extreme importance when strings are passed back to the application program. Here again, Window Manager will copy back to the application the number of characters specified by the length parameter. The string will still be null terminated so a scan of the string will result in finding its true value. The length specified must take into account room for a null terminator.

This means that the buffer reserved by the application to hold the returned string must be as long or longer than the specified length parameter. If it is not, the application will risk having its data area written over. This means if you have an allocated buffer 10 characters long and specify a length of 20, even if the string to be returned is only 5 characters long, 20 characters will be returned and 10 bytes of memory will be trashed.

If the length parameter is not specified large enough for the string to be returned, an error will occur and Window Manager will pass back the number of characters specified by the length parameter. The truncated string is still null terminated. This check is made only on the length parameter, not on the buffer itself.

Callable Routines

F.3 Table F-1 correlates generic procedure calls to the corresponding FORTRAN function calls. These are all the calls the application can make to Window Manager. For a summary of the calls that Window Manager makes to the application, see paragraph F.4, Routines Called by Window Manager. The table provides a specification of the FORTRAN calling sequence, a description of the function, and the function parameters. All routines are FORTRAN functions and must be declared and used as such.

Function Return Codes

F.3.1 All of the function codes return a status code. The returned value from each function is zero if no error was encountered or nonzero if an error condition occurred.

These nonzero codes are divided into two levels of severity. A negative code is a warning code. Warning codes are returned when the call did not complete as expected, but no harm was done. These codes can be ignored, but they indicate incorrect use of Window Manager and should be checked during the development process.

Positive codes indicate serious error conditions. When an application program detects a positive code, the problem must be identified and corrected. These positive codes mean that something required to process the call was in error, processing was halted, and future calls will probably fail as well.

A message for any of the nonzero codes can be displayed by calling the Display Message From Message Manager routine. The messages are stored in the LIERRMSG.NM\$ file. This is the filename that must be supplied in the Message Manager call along with the code returned from the Window Manager call. LIERRMSG.NM\$ is included on the Window Manager object disk.

The Add Window, Select Window, Refresh Window, Display Window, Release Window, and Delete Window operations cannot return a detailed status code when operating on all windows in a screen. To be more specific, when the window number parameter of these calls is set to -1 , the status code returned indicates that an error or warning was encountered during operations on one or more windows. This status implies a success or failure for the operations on all the windows. When the window number parameter is set to a value equal to or greater than 1, a nonzero return status is a specific error condition.

The only operation performed on pop-up windows when the window number is set to -1 is the Delete operation. All other operations on pop-up windows must be performed with an individual call for each pop-up window.

Table F-1 FORTRAN Interface Routines

Name: INITIALIZE WINDOW MANAGER
Call Sequence: **WMINIT**()
Description: Detects which machine the application is running on, initializes the video display attributes, installs keyboard mapping if needed, and loads the run-time version of the internal phrase file (NLXPHRAS.NM\$), if used. This routine **must** be called before any other Window Manager routines to ensure that Window Manager works correctly.
Parameters: None

Name: RESET WINDOW MANAGER
Call Sequence: **WMRSET**()
Description: Restores the values set by the WMINIT procedure. **Must** be called to ensure program terminates correctly.
Parameters: None

CAUTION: WMRSET must be called prior to program termination. If it is not called, a system crash is likely to occur when the next program is executed. This is because WMRSET restores any keyboard mapping invoked by WMINIT and resets the cursor and video attributes. In addition to calling WMRSET before normal program termination, the application program should provide a way to call WMRSET in the event of a critical MS-DOS error in order to prevent a crash. For details on how to trap the termination addresses, see the information on DOS Interrupts 22H and 23H in the MS-DOS Operating System manual.

Name: LOAD SCREEN FILE
Call Sequence: **WMLOAD**(SCREEN, PATHNAME, LENGTH)
Description: Loads the screen description from the pathname into memory. A screen must be loaded before any of its windows can be manipulated by Window Manager.
Parameters:
 SCREEN: Output. Equal to 0 — Load failed. Integer greater than or equal to 1 — Integer assigned by WM to identify screen.
 PATHNAME: Input. Character string that specifies the pathname of the screen description file.
 LENGTH: Input. Maximum length to be reserved for the pathname string.

Name: ADD WINDOW
Call Sequence: **WMWADD**(SCREEN, WINDOW)
Description: Makes a window known to Window Manager for subsequent operations.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Adds all windows for a screen, **except** pop-up windows. Integer greater than or equal to 1 — Window number to add.

Table F-1 FORTRAN Interface Routines (Continued)

Name:	SELECT WINDOW
Call Sequence:	WMWSEL (SCREEN, WINDOW)
Description:	Makes a window active and flags the window for repainting, which will be performed with the next Receive call.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Integer equal to -1 — Selects all windows for a screen, except pop-up windows. Integer greater than or equal to 1 — Window number to select.
<hr/>	
Name:	REFRESH WINDOW
Call Sequence:	WMWREF (SCREEN, WINDOW)
Description:	Flags the window for text and border repainting. The window is redisplayed the next time a Receive call is performed.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Integer equal to -1 — Refreshes all windows for a screen, except pop-up windows. Integer greater than or equal to 1 — Window number to refresh.
<hr/>	
Name:	DISPLAY WINDOW
Call Sequence:	WMWDIS (SCREEN, WINDOW)
Description:	Repaints the window text and border without delay. A Receive call is not required for the Display procedure to execute.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Integer equal to -1 — Displays all windows for a screen, except pop-up windows. Integer greater than or equal to 1 — Window number to display.

Table F-1 FORTRAN Interface Routines (Continued)

Name: RECEIVE FROM WINDOW
Call Sequence: **MMWRCV**(SCREEN, WINDOW, ITEM, KEY, ITEXT, LENGTH)
Description: Receives user responses from an active window. Window Manager retains control until an item is selected or until an undefined key is pressed. At that time Window Manager returns control to the application call routine.
Parameters:
 SCREEN: Input and output. The input integer is assigned by WMLOAD to identify the screen. The output integer specifies the screen of the window from which the response is received.
 WINDOW: Input and output. The input integer specifies the window from which user response is desired. The output integer specifies the window of the item selected or the window the cursor was in when an unknown key was pressed.
 ITEM: Input and output. The input integer specifies the item upon which the cursor is initially placed. The output integer specifies the item selected.
 KEY: Output. Code for the key pressed by the user.
 ITEXT: Output. Character string to receive text of the selected item.
 LENGTH: Input. Maximum length reserved for string (item text).

Name: RELEASE WINDOW
Call Sequence: **MMWREL**(SCREEN, WINDOW)
Description: Renders the window inactive and flags the window to have only text repainted. The window is redisplayed the next time a Receive call is performed.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Releases all active windows for a screen, **except** pop-up windows. Integer greater than or equal to 1 — Window number to release.

Name: DELETE WINDOW
Call Sequence: **MMWDEL**(SCREEN, WINDOW)
Description: Deletes the window from the screen and makes it unknown to Window Manager. If a window that covers other windows is deleted, the covered windows are flagged to have their borders and text repainted.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Deletes all windows for a screen, **including** pop-up windows. Integer greater than or equal to 1 — Window number to delete.

Table F-1 FORTRAN Interface Routines (Continued)

Name:	SAVE SCREEN
Call Sequence:	WMWSAV (SCREEN, PATHNAME, LENGTH)
Description:	Saves a screen to a file. This does not unload a screen from memory, and any operation can be performed on any window in that screen after the SAVE procedure has been completed. This is useful if windows have been modified and they need to be saved.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen file to be saved.
PATHNAME:	Input. String that specifies the pathname of the file where the screen is to be saved.
LENGTH:	Maximum length reserved for the pathname string.
<hr/>	
Name:	UNLOAD SCREEN
Call Sequence:	WMUNLD (SCREEN)
Description:	Releases the screen description from memory.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
<hr/>	
Name:	ADD ITEM
Call Sequence:	WMIADD (SCREEN, WINDOW, ITEM)
Description:	Adds an item to the end of the item table and increases the size of the item table by 1.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
ITEM:	Output. Item number of the added item.
<hr/>	
Name:	INSERT ITEM
Call Sequence:	WMIINS (SCREEN, WINDOW, ITEM)
Description:	Inserts a new item into the item table in front of the item position specified. All subsequent items are renumbered.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
ITEM:	Input. Position in the item table where the item is to be inserted.

Table F-1 FORTRAN Interface Routines (Continued)

Name:	DELETE ITEM
Call Sequence:	WMIDEL (SCREEN, WINDOW, ITEM)
Description:	Deletes an item from the item table. All subsequent items are renumbered.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
ITEM:	Input. Number of the item to be deleted. The item number must be greater than zero.
<hr/>	
Name:	CREATE ITEM TABLE
Call Sequence:	WMICRE (SCREEN, WINDOW, NBR)
Description:	Creates an item table with the specified number of items. An item table can be created during initial window definition or during program execution. An item table is created when the current number of items is 0. It manipulates the item table based on the relationship between NBR and the current number of items in the window as follows: <ul style="list-style-type: none">■ Current number of items equals 0 — Creates an item table with NBR of items.■ NBR less than current number of items — Deletes items (current number items minus NBR) from the end of the item table. This condition returns a warning.■ NBR greater than current number of items — Adds items (NBR minus current number of items) to the end of the item table.■ NBR equals 0 — Deletes the entire item table. This also returns a warning.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number in the screen description.
NBR:	Input. Desired number of items to be created for the window.
<hr/>	
Name:	CREATE WINDOW
Call Sequence:	MMWCRE (SCREEN, WINDOW)
Description:	This routine creates a new window and adds it to the screen specified. If an invalid screen number is specified, a new screen will be created.
Parameters:	
SCREEN:	Input and output. Input integer assigned by WMLOAD to identify the screen to which the new window should be added. The output integer specifies the screen number of the new screen created if the input value was an invalid screen.
WINDOW:	Output. Window number of the window which was created.

Table F-1 FORTRAN Interface Routines (Continued)

Name: GET ATTRIBUTE VALUE
Call Sequence: WMGETV(SCREEN, WINDOW, ITEM, FIELD, IVALUE)
Description: References the window structure and retrieves a value for the requested numeric attribute.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific window.
 WINDOW: Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
 ITEM: Input. Number of the item containing the attribute about which information is being requested. The value is ignored if the field is not associated with an item.
 FIELD: Input. Name of attribute constant desired. See Table F-3, Data Declarations for the FORTRAN Interface, for a listing and description of the valid attribute constant names.
 IVALUE: Output. Current value of the requested attribute.

Name: GET STRING VALUE
Call Sequence: WMGETS(SCREEN, WINDOW, ITEM, FIELD, STRING, LENGTH)
Description: References the window structure and retrieves the string value for the requested string attribute.
Parameters:
 SCREEN: Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
 WINDOW: Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
 ITEM: Input. Number of the item containing the attribute about which information is being requested. The value is ignored if the attribute is not associated with an item.
 FIELD: Input. This parameter uses one of the following attribute constant names:
 wlabel
 wshowf
 itlabl
 wmpath
 ittext
 See Table F-3 for a description of the valid attribute constant names.
 STRING: Output. Character variable to receive returned string.
 LENGTH: Input. Maximum length reserved for the string.

Table F-1 FORTRAN Interface Routines (Continued)

Name:	SET ATTRIBUTE VALUE
Call Sequence:	WMSETV(SCREEN, WINDOW, ITEM, FIELD, IVALUE)
Description:	References the window structure and sets the numeric attribute to the specified value.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Number of the item having the attribute about which information is being specified. The value is ignored if the attribute is not associated with an item.
FIELD:	Input. Name of the attribute constant desired. See Table F-3 for a listing and description of the valid attribute constant names.
IVALUE:	Input. Integer value for the specified attribute.

CAUTION: No validation checks are performed on the IVALUE parameter in the WMSETV call. This reduces the code size of the high-level language interface. It is assumed that the value is in the correct range for the attribute being set. The other parameters—Screen, Window, Item, and Field—are validated and error codes are returned if they are incorrect. See Section 3, Window Attributes, for the legal attribute values.

Name:	SET STRING VALUE
Call Sequence:	WMSETS(SCREEN, WINDOW, ITEM, FIELD, STRING, LENGTH)
Description:	References the window structure and sets a string attribute to the specified string.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Number of the item having the attribute for which information is being specified. The value is ignored if the field is not associated with an item.
FIELD:	Input. This parameter uses one of the following attribute constant names: wlabel wshowf itlabl wmpath ittext See Table F-3 for a listing and description of the valid attribute constant names.
STRING:	Input. Character string constant containing the defined text.
LENGTH:	Input. Length of the string being entered.

Table F-1 FORTRAN Interface Routines (Continued)

Name:	CLEAR SCREEN
Call Sequence:	WMCLRS()
Description:	Clears the text from the display area.
Parameters:	None
<hr/>	
Name:	DISPLAY MESSAGE FROM MESSAGE MANAGER
Call Sequence:	DISMSG(MSGNUM, VARTXT, TXTLEN, VARSEP, MSGFIL, LENGTH)
Description:	Displays the message associated with a message number from a message file and inserts the variable text, if any is given, into the message.
Parameters:	
MSGNUM:	Input. Integer message number of the message that is to be displayed. The message number can be either a status from NaturalLink or a message built by the software designer using Message Builder.
VARTXT:	Input. String that contains a maximum of three variable text substrings to be inserted in the actual text message by Message Manager. If there is no variable text, the string is a null string. Each variable text substring must be separated by the character specified in the separator parameter. The separator character used cannot be a part of the variable text itself. For example, using the tilde as a separator would give: "1st variable text ~ 2nd variable text ~ 3rd variable text".
TXTLEN:	Input. Length of the variable text string.
VARSEP:	Input. Integer indicating the ASCII character which is used as the variable text separator. For example, a decimal value of 126 is passed to indicate the tilde ~ character.
MSGFIL:	Input. String that contains the message file pathname from which the text is to be retrieved.
LENGTH:	Input. Maximum length reserved for the message file pathname.
<hr/>	
Name:	RESET NATURALLINK MEMORY AREA
Call Sequence:	WMFLSH()
Description:	This routine clears out the memory area used by the NaturalLink run time. Any screens loaded or other data in this memory area will be lost and must be reloaded before NaturalLink processing can continue.
Parameters:	None.

Routines Called by Window Manager

F.4 Table F-2, Application-Provided Routines, correlates generic procedure calls to the corresponding FORTRAN function calls. These routines are supplied by the application and are called by Window Manager. If the functionality provided by these calls is not desired, the dummy routines provided on the Window Manager Runtime Object disk can be used instead. All of the application-provided calls except the validation call and the input call can return error status codes to Window Manager. If the status returned is nonzero, Window Manager will return to the application program with the status code.

Table F-2 Application-Provided Routines

Name:	APPLICATION VALIDATION ROUTINE
Call Sequence:	APPVAL(SCREEN, WINDOW, ITEM, KEY, DATATYPE)
Description:	This routine is provided by the developer. Window Manager calls this routine when an edit field needs to be type-checked. It must return the status of the validation check. The defined statuses are as follows: -2 — Invalid/clear, type check failed, clear out edit field -1 — Invalid/no clear, type check failed, field untouched 0 — Valid, type check passed, continue 1 — Ignore, type check not done, return to edit field 2 — Exit, return to application immediately
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the window containing the item being validated.
ITEM:	Input and output. The input integer specifies the item to be validated. The output integer indicates which item the cursor should be placed on upon return to Window Manager.
KEY:	Input. Integer code for the key pressed by the user.
DATATYPE:	Input. Datatype assigned this item in the Set Item Attributes option in Screen Builder.
<hr/>	
Name:	APPLICATION DISPLAY WINDOW
Call Sequence:	APPDIS(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be displayed. This routine must display all aspects of the window except the window border.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to display.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table F-2 Application-Provided Routines (Continued)

Name:	APPLICATION RECEIVE WINDOW
Call Sequence:	APPRCV(SCREEN, WINDOW, ITEM, KEY, UDWNBR, XPOS, YPOS)
Description:	This routine is provided by the developer. Window Manager calls it when information must be received from a user-defined window. This routine must handle all user input for the given window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window from which to receive information.
ITEM:	Input and Output. The input integer specifies the item upon which the cursor should be initially placed. The output integer specifies the item selected.
KEY:	Output. Integer code for the key pressed by the user.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).
XPOS:	Output. This is the integer indicating the X character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.
YPOS:	Output. This is the integer indicating the Y character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.
<hr/>	
Name:	APPLICATION DELETE WINDOW
Call Sequence:	APPDEL(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be deleted. This routine must delete all nontext portions of the window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to be deleted.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table F-2 Application-Provided Routines (Continued)

Name:	APPLICATION DISPLAY MESSAGE
Call Sequence:	APPMSG(MSGSCR, MSGWND, KEY, MSGNBR, MSGTYP, SCREEN, WINDOW, ITEM)
Description:	This routine is provided by the developer. It is called when a user-defined message needs to be displayed. This routine must handle the display of the message, response from the user to the message, and the deletion of the nontext portion of the message from the screen.
Parameters:	
MSGSCR:	Input. Integer assigned by Window Manager to identify the screen containing the message window.
MSGWND:	Input. Window number of the user-defined message window.
KEY:	Output. Integer code for the key pressed by the user.
MSGNBR:	Input. Number of the message to be displayed. This is the number assigned to the message by the Message Builder utility.
MSGTYP:	Input. Integer indicating the type of message being displayed. The integer is one of the following values: 5 — Help 6 — Error 7 — Warning 8 — Please Note
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen containing the window the cursor was in when the HELP key was pressed. It is used only for Help messages.
WINDOW:	Input. Window number of the window the cursor was in when the HELP key was pressed. Used only for Help messages.
ITEM:	Input. Input integer specifies the item the cursor was on when the HELP key was pressed. A value of -1 is passed when window level help is being displayed. It is used only for Help messages.
<hr/>	
Name:	APPLICATION INPUT ROUTINE
Call Sequence:	APPINP()
Description:	This routine is provided by the developer. Window Manager calls it when input from the user is needed. It can be used to provide an alternative input method to the keyboard. This routine, if used, must still return valid key codes. A value of 0 must be returned if there is no key code ready to be returned.
Parameters:	None.

FORTRAN Compiling Requirements

F.5 Window Manager requires the use of common blocks and Include files. These are provided with the Window Manager package. Certain initialization steps for the common block must also be performed before making any Window Manager interface calls.

Common Blocks and Include Files

F.5.1 A common block containing attribute names for the WMSETV, WMGETV, WMSETS, and WMGETS Window Manager routines is provided to keep hard-coded constants out of the FORTRAN code. These constants are likely to change or at least expand with future releases of Window Manager. The common name is /WMCOM/ and can be included with an INCLUDE: WMFIELD.COM statement. In any procedures which call the Window Manager routines to use window attributes (for example, WMGETV, WMSETS, and so on), you must include WMFIELD.COM as well as in the main program. The common block is initialized by the WMCINI.FOR routine included in this package. The external declarations of the Window Manager routines are in WMEXTRNS.FOR and should also be included in FORTRAN source modules. A listing of the common block is given in Table F-3.

Initialization of Window Manager

F.5.2 An application program needs to call the WMCINI routine (CALL WMCINI) to initialize the /WMCOM/ common block before the application program calls a WMGETV, WMGETS, WMSETV, or WMSETS routine.

Table F-3 Data Declarations for the FORTRAN Interface

<i>Name</i>	<i>Type</i>	<i>Value</i>	<i>Description for Nonzero or True Condition</i>
c			
c	Window position:		
c	wposux	int 1	Left-most column
c	wposuy	int 2	Top row
c	wposlx	int 3	Right-most column
c	wposly	int 4	Bottom row
c			
c	Window format:		
c	awdtyp	int 5	Window type
c	wincol	int 38	Number of columns
c	nwprio	int 10	Window priority (for application use only)
c	nwmusl	int 11	Multiple selection window
c	nwpopu	int 12	Pop-up window
c	inpant	int 46	Special repaint on receive
c	wofset	int 21	First item to be displayed (1 based)
c	iautsc	int 40	Show last item when painted
c	icentr	int 45	Center all items
c	inorep	int 43	Don't redisplay current items
c	inomul	int 42	Disable multiple line items
c	cursin	int 74	Allow cursor to enter window
c	idirct	int 44	Multiple column order
c	ilmlen	int 41	Maximum item label length
c	iljust	int 47	Item label justification
c	wshowf	str 20	Pathname of window file
c	nwactv	int 13	Active window (read only access)
c	witmnb	int 39	Number of items (read only access)
c			
c	Active window attributes:		
c	ablink	int 6	Blinking
c	aunder	int 7	Underline
c	arever	int 8	Reverse video
c	actint	int 9	Intensity/color
c			
c	Inactive window attributes:		
c	nwblnk	int 16	Blinking
c	nwundr	int 17	Underline
c	nwrevr	int 18	Reverse video
c	nwintn	int 19	Intensity/color
c			
c	Window Label attributes:		
c	wlabel	str 22	Window label
c	wlvsbl	int 23	Invisible label
c	wlpos	int 24	Label position
c	wlcent	int 25	Centered label
c	wlblnk	int 27	Blinking label
c	wlundr	int 28	Underlined label
c	wlrevr	int 29	Reverse video label
c	wlint	int 30	Label intensity/color
c	wlaint	int 26	Use item intensity

Table F-3 Data Declarations for the FORTRAN Interface (Continued)

<i>Name</i>	<i>Type</i>	<i>Value</i>	<i>Description for Nonzero or True Condition</i>
c			
c Cursor attributes:			
c	crsize	int 31	Cursor size
c	cblink	int 33	Blinking cursor
c	cunder	int 34	Underlined cursor
c	crever	int 35	Reverse video cursor
c	curint	int 37	Cursor intensity/color
c	calint	int 32	Use Item intensity
c	cstays	int 36	Cursor remains on item if receive abort
c			
c Border attributes:			
c	nwbord	int 14	Bordered window
c	bdrtspace	int 72	Border takes up space (0 = Yes)
c	nwscmk	int 15	Display scroll markers
c	bdrrv	int 73	Reverse video border
c	bdrint	int 74	Border intensity/color
c			
c Item format attributes:			
c	ittext	str 66	Item text
c	itchos	int 56	Chosen/enable attributes
c	itvsbl	int 63	Visible item
c	itunsl	int 59	Unselectable item
c	itdspl	int 55	Displayed
c	itmlen	int 48	Maximum edit field length
c	itreqr	int 57	Required edit field
c	itecho	int 58	Echo edit field input
c	dattyp	int 65	Edit field datatype
c			
c Item-chosen attributes:			
c	itblnk	int 60	Blinking chosen item
c	itundr	int 61	Underlined chosen item
c	itrevr	int 62	Reverse video chosen item
c	itint	int 64	Chosen item intensity/color
c			
c Item label attributes:			
c	itlabl	str 54	Item label
c	ilblnk	int 50	Blinking label
c	ilundr	int 51	Underlined label
c	ilrevr	int 52	Reverse video label
c	ilint	int 53	Label intensity/color
c	ilaint	int 49	Use item intensity
c			
c Other miscellaneous attributes:			
c	wmpath	int 71	Window Manager help/phrase file path
c	pctype	int 75	Machine type (read only access)
c	lcdon	int 76	PRO-LITE lcd screen indicator

Memory Considerations

F.6 Since the NaturalLink Window Manager and the FORTRAN application program coexist in memory, sufficient memory to accommodate the code and data areas of each is required.

To achieve correct ordering of the code and data areas for the FORTRAN application program and Window Manager object, a routine called LIMSFSH has been added. This routine defines the memory organization. It places the application program code in low memory, followed by the Window Manager code, the Window Manager data area, and finally the application program's data area. The routine declares the Window Manager stack and heap size, which must be statically allocated. Since the application program's stack and heap area occurs last, it can expand up to the physical memory limits of the machine.

Get-memory errors appear when the Window Manager heap management routines run out of memory. The only way to prevent these errors, if they occur, is to increase the size of the Window Manager heap area or to reduce the number of screens (or screen sizes) loaded into memory at the time the error occurred.

The default stack and heap sizes for Window Manager are as follows:

2K stack = hexadecimal 800 bytes
32K heap = hexadecimal 8000 bytes

The maximum size of the Window Manager stack and heap is the size in bytes determined by the difference of hexadecimal FFF0 and the size of the Window Manager data segment (called NL_DATA and found in the application program's link map). At least hexadecimal 800 bytes should be allocated for the Window Manager stack size. Using this minimum recommendation for the stack size, the maximum amount of memory that can be allocated for the heap area is hexadecimal F7F0 minus the size of the data segment NL_DATA. This will be approximately hexadecimal ED00 bytes for application programs that use only the Window Manager object and slightly less for those that additionally use Natural Language object. A change in the Window Manager stack and heap sizes will require changing the LIMSFSH.ASM source file, reassembling the routine, and relinking the application.

Linking Considerations

F.7 An example of a link control file used to link a simple FORTRAN application program follows. Most of the NaturalLink runtime has been placed in libraries. The order of the libraries is significant and they must be placed in the order shown.

LIMSFSH +	Memory organization module. This must occur first.
mymain +	The FORTRAN application's main program.
WMCINI +	FORTRAN subroutine to initialize WMCOM.
...	Any application subroutines, if used.
demo /M	The .EXE file.
demo.MAP	The .MAP file.
mylib +	Any application libraries, if used.
LIMSF +	Library of language interface routines to Window Manager for FORTRAN.
WM +	Window Manager object library.
APPOTH +	Dummy Window Manager called application routines.
FORTTRAN	FORTTRAN run-time library.

Dummy Routines Library

F.7.1 The APPOTH library contains dummy routines for all application routines which are called by Window Manager. These include APPVAL, APPRCV, APPDIS, APPDEL, APPMSG, and APPINP. If you do not use the features provided by one or more of these routines, the reference to them will be resolved by the dummy modules in this library. If you have your own versions of these routines, they must be explicitly linked; if they are in a library, the library must come before APPOTH in the link stream. If you create any libraries of your application routines, these libraries must come before any NaturalLink libraries in the link stream.

WMKEYDEF.OBJ and WMSTRDEF.OBJ Files

F.7.2 The key definition file (WMKEYDEF.OBJ) and the internal phrase file (WMSTRDEF.OBJ) are included as part of the Window Manager library (WM.LIB). If you have created your own version of these files (by using the KBUILD utility for WMKEYDEF or by reassembling the source file for WMSTRDEF), you must explicitly link your own version. Your version of the files should be placed in the link stream following any application subroutines that you may have. The linker will then use your versions instead of the files in the library.

Object Code Segments

F.7.3 The object code for Window Manager is divided into two groups. Segments with the name NL_PROG contain executable code, while segments with the name NL_DATA contain static data, heap, and the Window Manager run-time stack.

Restrictions

F.8 The maximum number of screens loaded from files is set to 20. A screen can have up to 255 windows and each window can have up to 65,536 items. Since screens are stored in the Window Manager heap, a screen which contains excessive windows and items can cause a get-memory error when loaded.

COMPILED BASIC INTERFACE



<i>Paragraph</i>	<i>Title</i>	<i>Page</i>
G.1	Introduction	G-3
G.2	Parameter Characteristics	G-3
G.2.1	Zero-Base Values	G-3
G.2.2	Names of Variables	G-3
G.2.3	Integers	G-3
G.2.4	Character Strings	G-3
G.2.4.1	Passing Strings to Window Manager	G-4
G.2.4.2	Strings Returned by Window Manager	G-4
G.3	Callable Routines	G-5
G.3.1	Procedure Return Codes	G-5
G.4	Routines Called by Window Manager	G-15
G.5	BASIC Compiling Requirements	G-22
G.5.1	Include Files	G-22
G.6	Memory Considerations	G-25
G.7	Linking Considerations	G-26
G.7.1	Dummy Routines Library	G-26
G.7.2	WMKEYDEF.OBJ and WMSTRDEF.OBJ Files	G-26
G.7.3	Object Code Segments	G-26
G.8	Restrictions	G-26

Introduction

G.1 This appendix presents the information and procedures required to use the NaturalLink Window Manager with an MS-Compiled BASIC application program. Generally, the parts of the NaturalLink package referenced from Compiled BASIC procedures are linked with the Compiled BASIC application program and act as an application program extension.

Parameter Characteristics

G.2 The only types of parameters used in the Window Manager call routines are integers and character strings. The exact order of these parameters is shown in Table G-1, Compiled BASIC Interface Routines.

Zero-Base Values

G.2.1 For all Compiled BASIC Window Manager calls that use a screen, window, or item identifier, the value of these identifiers is zero-base relative. The smallest possible screen, window, or item number is 0. This corresponds to window 0 and item 0 as assigned by Screen Builder.

Names of Variables

G.2.2 Certain variable names commonly used in calls to Window Manager routines are *not* applicable when writing in Compiled BASIC because they conflict with Compiled BASIC reserved words such as KEY and SCREEN. In the examples in Table G-3, Data Declarations for the Compiled BASIC Interface, these words have been replaced with KEY1 and SCREEN1 to minimize the differences from the examples in the appendixes for other high-level language interfaces.

Integers

G.2.3 Integers passed from Compiled BASIC to Window Manager should be defined as INTEGERS, using either the DEFINT command or the integer override % character.

Character Strings

G.2.4 Window Manager strings are stored as eight-bit ASCII characters in sequential bytes of memory. A string pointer points to the first character, and the string is delimited by a null character (0 binary).

When passing strings from BASIC to Window Manager and back, only one argument is required. Use the name of a string variable, such as FILENAME\$.

***Passing Strings
to Window Manager***

G.2.4.1 When passing strings to Window Manager, a length parameter should *not* be specified because the BASIC string descriptor contains the length allocated for the string. Note that this differs from other high-level languages that require a length parameter for strings.

The first example illustrates the passing of a character string as a constant.

```
CALL WMSETS(SCREEN1,WIND,ITEM,FIELD,"sample string",STATUS)
```

The second example illustrates the passing of a string variable. Note that the assignment allocates memory for the string.

```
STRVAL$="sample string"  
CALL WMSETS(SCREEN1,WIND,ITEM,FIELD,STRVAL$,STATUS)
```

Compiled BASIC passes the address of the string descriptor to the WMSETS (Set String Value) call. Window Manager uses the length contained in the string descriptor to indicate how many characters need to be copied from the BASIC application's data area to the Window Manager data area. Window Manager allocates the number of characters specified by the length plus one character for a null terminator, which is always appended to the string.

In both examples, 13 characters are copied.

***Strings Returned
by Window Manager***

G.2.4.2 For strings passed from Window Manager to BASIC, the maximum length of the string must be allocated. One way to do this is shown in the following example:

```
STRVAL$ = SPACE$(maxLength)
```

The length contained in the string descriptor as a result of this allocation is of extreme importance when strings are passed back to the application program. Window Manager will copy back the number of characters specified by this length to the application.

Window Manager will place an ASCII null byte at the end of the string as a terminator. When you return from a Window Manager routine, you can use the following command to correct the length of the string descriptor, which indicates the position of the null character.

```
STRVAL$=LEFT$(STRVAL$,INSTR(STRVAL$,CHR$(0))).
```

To test whether a null string was passed from Window Manager, use the following command sequence:

```
STRVAL$=LEFT$(STRVAL$, INSTR(STRVAL$, CHR$(0))).  
if STRVAL$=CHR$(0) goto XXX
```

The allocated string must take into account room for the null terminator. If the allocated string buffer is not large enough for the string to be returned, an error will occur and Window Manager will pass back only the number of characters specified by the length contained in the string descriptor. The truncated string is still null terminated.

Callable Routines

G.3 Table G-1, Compiled BASIC Interface Routines, correlates generic procedure calls to the corresponding Compiled BASIC function calls. These are the calls that the application can make to Window Manager. The table provides a specification of the Compiled BASIC calling sequence, a description of the procedure, and the procedure parameters. All routines are external subroutines and must be invoked using the CALL statement. For a summary of the calls that Window Manager makes to the application, see paragraph G.4, Routines Called by Window Manager.

Procedure Return Codes

G.3.1 All the procedures return a status code. The status is always sent as the last parameter in the list for every call, regardless of the number of parameters sent. This is because Compiled BASIC does not support external function calls. This last parameter is treated the same as a return function value in the other high-level language interfaces.

The returned value from each procedure is zero if no error or warning was encountered or nonzero if an error or warning condition occurred.

These nonzero codes are divided into two levels of severity: negative codes and positive codes. A negative code is a warning code. Warning codes are returned when the call did not complete as expected but no harm was done. These codes can be ignored; they indicate incorrect or inefficient use of Window Manager and should be checked during the development process.

Positive codes indicate serious error conditions. When an application program detects a positive code, the problem must be identified and corrected. These positive codes mean that something required to process the call was in error, processing was halted, and future calls will probably fail as well.

A message for any of the nonzero codes can be displayed by calling the Display Message From Message Manager routine. The messages are stored in the LIERRMSG.NM\$ file. This is the filename that must be supplied in the Message Manager call, along with the code returned from the Window Manager call. LIERRMSG.NM\$ is included on the Window Manager object disk. A listing of all error codes and messages can be found in Appendix H, Window Manager Error Codes.

The Add Window, Select Window, Refresh Window, Display Window, Release Window, and Delete Window operations cannot return a detailed status code when operating on all windows in a screen. To be more specific, when the window number parameter of these calls is set to -1, the status code returned indicates that an error or warning was encountered during operations on one or more windows. This status implies a success or failure for the operations on all the windows. When the window number parameter is set to a value equal to or greater than 0, a nonzero return status is a specific error condition.

The only operation performed on pop-up windows when the window number is set to -1 is the Delete operation. All other operations on pop-up windows must be performed with an individual call for each pop-up window.

Table G-1 Compiled BASIC Interface Routines

Name: INITIALIZE WINDOW MANAGER
 Call Sequence: CALL WMINIT(STATUS)
 Description: Detects which machine the application is running on, initializes the video display attributes, installs keyboard mapping if needed, and loads the run-time version of the internal phrase file (NLXPHRAS.NM\$), if used. This routine **must** be called before any other Window Manager routines to ensure that Window Manager works correctly.
 Parameters:
 STATUS: Output. Call return status.

Name: RESET WINDOW MANAGER
 Call Sequence: CALL WMRSET(STATUS)
 Description: Restores the values set by the WMINIT procedure. It **must** be called to ensure program terminates correctly.
 Parameters:
 STATUS: Output. Call return status.

CAUTION: WMRSET must be called prior to program termination. If it is not called, a system crash is likely to occur when the next program is executed. This is because WMRSET restores any keyboard mapping invoked by WMINIT, and resets the cursor and video attributes. In addition to calling WMRSET before normal program termination, the application program should provide a way to call WMRSET in the event of a critical MS-DOS error to prevent a crash. For details on how to trap the termination addresses, see the information on DOS Interrupts 22H and 23H in *The MS-DOS Operating System* manual.

Table G-1 Compiled BASIC Interface Routines (Continued)

Name: LOAD SCREEN FILE

Call Sequence: CALL WMLOAD(SCREEN1, PATHNAME\$, STATUS)

Description: Loads the screen description from the pathname into memory. A screen must be loaded before any of its windows can be manipulated by Window Manager.

Parameters:

SCREEN1: Output. Equal to -1 — Load failed. Integer greater than or equal to 0 — Integer assigned by Window Manager to identify the screen.

PATHNAME\$: Input. Character string that specifies the pathname of the screen description file.

STATUS: Output. Call return status.

Name: ADD WINDOW

Call Sequence: CALL WMWADD(SCREEN1, WINDOW, STATUS)

Description: Makes a window known to Window Manager for subsequent operations.

Parameters:

SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.

WINDOW: Input. Integer equal to -1 — Adds all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to add.

STATUS: Output. Call return status.

Name: SELECT WINDOW

Call Sequence: CALL WMWSEL(SCREEN1, WINDOW, STATUS)

Description: Makes a window active and flags the window for repainting, which will be performed with the next Receive call.

Parameters:

SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.

WINDOW: Input. Integer equal to -1 — Selects all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to select.

STATUS: Output. Call return status.

Table G-1 Compiled BASIC Interface Routines (Continued)

Name: REFRESH WINDOW
Call Sequence: **CALL WMWREF**(SCREEN1, WINDOW, STATUS)
Description: Flags the window for text and border repainting. The window is redisplayed the next time a Receive call is performed.
Parameters:
 SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Refreshes all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to refresh.
 STATUS: Output. Call return status.

Name: DISPLAY WINDOW
Call Sequence: **CALL WMWDIS**(SCREEN1, WINDOW, STATUS)
Description: Repaints the window text and border without delay. A Receive call is not required for the Display procedure to execute.
Parameters:
 SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Integer equal to -1 — Displays all windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to display.
 STATUS: Output. Call return status.

Name: RECEIVE FROM WINDOW
Call Sequence: **CALL MMWRCV**(SCREEN1, WINDOW, ITEM, KEY1, ITEXT\$, STATUS)
Description: Receives user responses from an active window. Window Manager retains control until an item is selected or until an undefined key is pressed, then returns control to the application call routine.
Parameters:
 SCREEN1: Input and output. The input integer is assigned by WMLOAD to identify the screen. The output integer specifies the screen of the window from which the response is received.
 WINDOW: Input and output. The input integer specifies the window from which user response is desired. The output integer specifies the window of the item selected or the window the cursor was in when an unknown key was pressed.
 ITEM: Input and output. The input integer specifies the item upon which the cursor is initially placed. The output integer specifies the item selected.
 KEY1: Output. Code for the key pressed by the user.
 ITEXT\$: Output. Character string to receive text of the selected item. Space must be allocated for this string before making the call.
 STATUS: Output. Call return status.

Table G-1 Compiled BASIC Interface Routines (Continued)

Name: RELEASE WINDOW
Call Sequence: CALL WMWREL(SCREEN1, WINDOW, STATUS)
Description: Renders the window inactive and flags the window to have only text repainted. The window is redisplayed the next time a Receive call is performed.
Parameters:
SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
WINDOW: Input. Integer equal to -1 — Releases all active windows for a screen, **except** pop-up windows. Integer greater than or equal to 0 — Window number to release.
STATUS: Output. Call return status.

Name: DELETE WINDOW
Call Sequence: CALL MMWDEL(SCREEN1, WINDOW, STATUS)
Description: Deletes the window from the screen and makes it unknown to Window Manager. If a window that covers other windows is deleted, the covered windows are flagged to have their borders and text repainted.
Parameters:
SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
WINDOW: Input. Integer equal to -1 — Deletes all windows for a screen, **including** pop-up windows. Integer greater than or equal to 0 — Window number to delete.
STATUS: Output. Call return status.

Name: SAVE SCREEN
Call Sequence: CALL MMWSAV(SCREEN1, PATHNAME\$, STATUS)
Description: Saves a screen to a file. This does not unload a screen from memory, and any operation can be performed on any window in that screen after the SAVE procedure has been completed. This is useful if windows have been modified and need to be saved.
Parameters:
SCREEN1: Input. Integer assigned by WMLOAD to identify the screen file to be saved.
PATHNAME\$: Input. String that specifies the pathname of the file where the screen is to be saved.
STATUS: Output. Call return status.

Table G-1 Compiled BASIC Interface Routines (Continued)

Name: UNLOAD SCREEN
Call Sequence: CALL WMUNLD(SCREEN1, STATUS)
Description: Releases the screen description from memory.
Parameters:
 SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
 STATUS: Output. Call return status.

Name: ADD ITEM
Call Sequence: CALL WMIADD(SCREEN1, WINDOW, ITEM, STATUS)
Description: Adds an item to the end of the item table and increases the size of the item table by 1.
Parameters:
 SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Output. Item number of the added item.
 STATUS: Output. Call return status.

Name: INSERT ITEM
Call Sequence: CALL WMIINS(SCREEN1, WINDOW, ITEM, STATUS)
Description: Inserts a new item into the item table in front of the item position specified. All subsequent items are renumbered.
Parameters:
 SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Input. Position in the item table where the item is to be inserted.
 STATUS: Output. Call return status.

Name: DELETE ITEM
Call Sequence: CALL WMIDEL(SCREEN1, WINDOW, ITEM, STATUS)
Description: Deletes an item from the item table. All subsequent items are renumbered.
Parameters:
 SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.
 WINDOW: Input. Window number in the screen description.
 ITEM: Input. Number of item to be deleted. The item number must be greater than zero.
 STATUS: Output. Call return status.

Table G-1 Compiled BASIC Interface Routines (Continued)

Name: CREATE ITEM TABLE

Call Sequence: CALL WMICRE(SCREEN1, WINDOW, NBR, STATUS)

Description: Creates an item table with the specified number of items. An item table can be created during initial window definition or during program execution. An item table is created when the number of items is 0. Manipulates the item table based on the relationship between NBR and the current number of items in the window as follows:

- Current number of items equals 0 — Creates an item table with NBR of items.
- NBR less than current number of items — Deletes items (current number items minus NBR) from the end of the item table. This condition returns a warning.
- NBR greater than current number of items — Adds items (NBR minus current number of items) to the end of the item table.
- NBR equals 0 — Deletes the entire item table. This also returns a warning.

Parameters:

SCREEN1: Input. Integer assigned by WMLOAD to identify the screen.

WINDOW: Input. Window number in the screen description.

NBR: Input. Desired number of items in the table.

STATUS: Output. Call return status.

Name: CREATE WINDOW

Call Sequence: CALL WMWCRE(SCREEN1, WINDOW, STATUS)

Description: This routine creates a new window and adds it to the screen specified. If an invalid screen number is specified, a new screen will be created.

Parameters:

SCREEN1: Input and output. Input integer assigned by WMLOAD to identify the screen to which the new window should be added. The output integer specifies the screen number of the new screen created if the input value was an invalid screen.

WINDOW: Output. Window number of the window which was created.

STATUS: Output. Call return status.

Table G-2**Application-Provided Routines (Continued)**

Name:	SET ATTRIBUTE VALUE
Call Sequence:	CALL WMSETV(SCREEN1, WINDOW, ITEM, FIELD, IVALUE, STATUS)
Description:	References the window structure and sets the numeric attribute to the specified value.
Parameters:	
SCREEN1:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Number of the item having the attribute about which information is specified. The value is ignored if the attribute is not associated with an item.
FIELD:	Input. Name of the attribute constant desired. See Table G-3 for a listing and description of the valid attribute constant names.
IVALUE:	Input. Integer value for the specified attribute.
STATUS:	Output. Call return status.

CAUTION: No validation checks are performed on the IVALUE parameter in the WMSETV call. This reduces the code size of the high-level language interface. It is assumed that the value is in the correct range for the attribute being set. The other parameters — SCREEN1, WINDOW, ITEM, and FIELD — are validated, and error codes are returned if they are incorrect. See Section 3, Window Attributes, for the legal attribute values.

Name:	SET STRING VALUE
Call Sequence:	CALL WMSETS(SCREEN1, WINDOW, ITEM, FIELD, STRVAL\$, STATUS)
Description:	References the window structure and sets a string attribute to the specified string.
Parameters:	
SCREEN1:	Input. Integer assigned by WMLOAD to identify the screen. The value is ignored if the field is not associated with a specific screen.
WINDOW:	Input. Window number in the screen description. The value is ignored if the field is not associated with a specific window.
ITEM:	Input. Number of the item having the attribute for which information is specified. The value is ignored if the field is not associated with an item.
FIELD:	Input. This parameter has one of the following attribute constant names: Llwlabel Llwshowf Llittext Llitlabl Llwmpath See Table G-3 for a listing and description of the valid attribute constant names.
STRVAL\$:	Input. Character string constant containing the defined text.
STATUS:	Output. Call return status.

Table G-1 Compiled BASIC Interface Routines (Continued)

Name: CLEAR SCREEN
Call Sequence: CALL WMCLRS(STATUS)
Description: Clears the text from the display area.
Parameters: None
STATUS: Output. Call return status.

Name: DISPLAY MESSAGE FROM MESSAGE MANAGER
Call Sequence: CALL DISMSG(MSGNUM, VARTXT\$, VARSEP, MSGFIL\$, STATUS)
Description: Displays the message associated with a message number from a message file and inserts the variable text (if any is given) into the message.
Parameters:
MSGNUM: Input. Integer message number of the message that is to be displayed. The message number can be either a status from NaturalLink or a message built by the software designer using Message Builder.
VARTXT\$: Input. String containing a maximum of three variable text substrings to be inserted in the actual text message by Message Manager. If there is no variable text, the string is a null string. Each variable text substring must be separated by the character specified in the separator parameter. The separator character used cannot be a part of the variable text itself. For example, using the tilde as a separator would give: "1st variable text ~ 2nd variable text ~ 3rd variable text".
VARSEP: Input. Integer indicating the ASCII character which is used as the variable text separator. For example, a decimal value of 126 is passed to indicate the tilde ~ character.
MSGFIL\$: Input. String that contains the message file pathname from which the text is to be retrieved.
STATUS: Output. Call return status.

Name: RESET NATURALLINK MEMORY AREA
Call Sequence: CALL WMFLSH(STATUS)
Description: This routine clears out the memory area used by the NaturalLink run time. Any screens loaded or other data in this memory area will be lost and must be reloaded before NaturalLink processing can continue.
Parameters:
STATUS: Output. Call return status.

Routines Called by Window Manager

G.4 The following is an example for coding assembly language routines called by the NaturalLink software (Window Manager and Sessioner). This code example is not meant to be a complete working routine. It is intended only to show the important points you need to know when writing your assembly language routine. Following the code are paragraphs that begin with numbers referring to the numbers in the example code. The numbered paragraphs discuss coding details.

```
(1)      NAME APPVAL
(2)      PUBLIC APPVAL

(3)      EXTRN WMSETS:FAR
(3)      EXTRN APPDIS:FAR

(4) DGROUP GROUP      DATA
(4) DATA  SEGMENT    BYTE      PUBLIC  'DATA'
SCREEN    DW  ?      ; parameter passed to APPVAL by WMWRCV
WINDOW    DW  ?      ; window number
ITEM      DW  ?      ; item number
TEMP      DW  ?      ; for passing offset of immediate variables
STRING1   DB  'text for string2'
STRINGPTR DW  ?      ; string descriptor: first word is length,
STRINGOFF DW  ?      ; second word contains offset from DS.
STAT      DW  ?      ; WMSETS function return code
MODITEM   DW  ?      ; parameter passed to GIVEPARMS from BASIC main
RETCODE   DW  ?      ; APPVAL return code
LIittext  EQU 65     ; item text field constant for WMSETS call
(4) DATA  ENDS

(5) VALSEG SEGMENT    'VALSEG'
(6)        ASSUME    CS:VALSEG,DS:DGROUP,ES:DGROUP,SS:NOTHING

        UDWNBR     DW  ?
(7) APPVAL  PROC FAR
        PUSH       BP
        MOV        BP,SP
(8)        MOV        BX,[BP+16]      ;save input parameter
        MOV        SCREEN, BX
(9)        MOV        AX,OFFSET DGROUP:SCREEN
        PUSH       AX
        MOV        AX,OFFSET DGROUP:WINDOW
        PUSH       AX
        MOV        AX,OFFSET DGROUP:ITEM
        PUSH       AX
(10)       MOV        AX,LIittext
        MOV        TEMP,AX
        MOV        AX,OFFSET DGROUP:TEMP
        PUSH       AX
        MOV        AX,OFFSET DGROUP:STRING1
(11)       MOV        STRINGOFF,AX
(11)       MOV        STRINGPTR,12
        MOV        AX,OFFSET DGROUP:STRINGPTR
        PUSH       AX
        MOV        AX,OFFSET DGROUP:STAT
        PUSH       AX
(12)       CALL      WMSETS
```

```

                MOV     AX,SCREEN
                PUSH   AX
                MOV     AX,WINDOW
                PUSH   AX
(13)           MOV     AX,CS:UDWNR ; user-defined window number
                PUSH   AX
                CALL   APPDIS
(14)           POP     CX
                POP     CX
                POP     CX
                PUSH   ES
(15)           LES     BX,[BP+10] ; rewrite item passed by reference
                MOV     AX,MODITEM
                MOV     ES:[BX],AX
                POP     ES
(16)           MOV     AX,RETCODE
                POP     BP
                RET
(7) APPVAL    ENDP

(17) GIVEPARMS PROC FAR
                PUSH   BP
                MOV     BP,SP
                MOV     BX,[BP+8] ; get value from BASIC main program
                MOV     AX,[BX] ; make value known for APPVAL
                MOV     MODITEM,AX
                POP     BP
                RET     2*2
                GIVEPARMS ENDP

(5) VALSEG    ENDS
                END

```

(1), (2), (5), (6), (7) — These statements must be in your assembly routine. You will have to replace the routine name and segment name, depending on the routine for which you are writing the code. If coding the APPRCV routine, replace APPVAL with APPRCV and VALSEG with RCVSEG.

(3) — You must declare any NaturalLink routines that will be called from your assembly language routine to be EXTRN:FAR. If calling WMSETS from the APPVAL routine, the WMSETS routine must be declared EXTRN:FAR. Also, any assembly language routines outside of the VALSEG that are called from your APPVAL routine must also be declared as EXTRN:FAR.

(4) — These statements must be in your assembly routine as written. Note that this defines the variables to be in the shared DATA data segment. The total data area shared by the BASIC main program and all the assembly language routines called by NaturalLink is 64K bytes.

(5) — The name of your segment will always be the last three letters of the routine name, followed by SEG. See the LIMSBSH.ASM source file included on the Window Manager Run-Time Object diskette for the segment names of all the application assembly language routines called by NaturalLink. Having each routine in a separate segment gives you up to 64K bytes of code space for each routine. Since overlaying is not possible for the BASIC high-level language interface, you can circumvent this restriction by calling other assembly language routines defined in different segments.

(6) — This statement must be in your assembly routine. You must assume CS to be the name of the segment discussed in (5), and assume DS to be DGROUP.

(8) — Input (value) parameters will be one word (16 bits) and output (address) parameters will be a long word (32 bits). Since a far call was made by NaturalLink to this routine, after pushing BP the first parameter can be accessed by [BP + 6]. Since the parameters are pushed in reverse order, the parameter at [BP + 6] will be the last parameter shown in the calling sequence. In the case of the APPVAL routine, this would be the datatype parameter, followed by the KEY parameter at [BP + 8]. The item parameter for the APPVAL routine is an output parameter; thus, because it is a long word, it can be accessed by [BP + 10] and [BP + 12]. Finally, the window and screen parameters are at [BP + 14] and [BP + 16], respectively.

(9) — All parameters to NaturalLink calls must reside in and be pushed as OFFSETs from DGROUP. All parameters must push addresses of variables, not values, to maintain compatibility with the way the CALL statement compiles in the BASIC main program.

(10) — EQU's do not need a segment override prefix, but since an offset must be passed, load the constant into a memory location and pass the offset of that memory location.

(11) — When passing a string to a NaturalLink routine, use a two-word string pointer. Move the length of the string (or the maximum size of the string buffer when sending a parameter to be written to by NaturalLink) into the first word of the pointer. Move the offset of the string buffer into the second word of the string pointer. Push the address (the offset only) of the string pointer as the parameter. Note that there is no separate length parameter as in the other languages, because the length is included in the string pointer.

(12) — When calling NaturalLink routines (such as WMLOAD, WMWADD, and so on) from your assembly language routine, you should *not* pop the parameters off the stack after returning from the call. This is automatic, since the NaturalLink software is coded in C.

(13) — To save space in the BASIC data segment, you can use variables declared in your CS by using a CS: override prefix. However, all parameters passed to NaturalLink **must** reside in DGROUP.

(14) — If calling assembly language routines normally called by NaturalLink (such as APPMSG, APPRCV, and so on), you must pop the parameters after the call. Do *not* pop the parameters in the called routine because, when that routine is called by NaturalLink, the popping of the stack is handled by the high-level language interface.

(15) — Output parameters are two words; therefore, a segment and an offset must be used.

(16) — The function return code must be put into the AX register before returning from the routine.

(17) — This is an example routine that will let you access and modify variables from your BASIC main program in your assembly language routine. The call in your BASIC main program is as follows:

```
CALL GIVEPARMS(MODITEM)
```

The parameters in the call are the variables in your assembly language routine. This call must be made before NaturalLink calls the application assembly language routine.

NOTE: The application assembly language routines called by NaturalLink (that is, APPVAL, APPRCV, and so on) should not be called from your BASIC main program. They will not work correctly, and undesirable results may occur.

Table G-2, Application-Provided Routines, correlates generic procedure calls to the corresponding C function calls. These routines are supplied by the application and are called by Window Manager. If the functionality provided by these calls is not desired, the dummy library (APPOTH) provided on the Window Manager Run-Time Object diskette can be used instead. All of the application-provided calls, except the validation call and the input call, can return error status codes to Window Manager. If the status returned is nonzero, Window Manager will return to the application program with the status code.

BASIC does not provide the ability to define functions that can be called by external routines. Thus all called routines that the application must provide must be written in assembly language.

Table G-2 Application-Provided Routines

Name:	APPLICATION VALIDATION ROUTINE
Call Sequence:	APPVAL(SCREEN, WINDOW, ITEM, KEY, DATATYPE)
Description:	This routine is provided by the developer. Window Manager calls this routine when an edit field needs to be type-checked. It must return the status of the validation check. The defined statuses are as follows: -2 — Invalid/clear, type check failed, clear out edit field -1 — Invalid/no clear, type check failed, field untouched 0 — Valid, type check passed, continue 1 — Ignore, type check not done, return to edit field 2 — Exit, return to application immediately
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the window containing the item being validated.
ITEM:	Input and output. The input integer specifies the item to be validated. The output integer indicates which item the cursor should be placed on upon return to Window Manager.
KEY:	Input. Integer code for the key pressed by the user.
DATATYPE:	Input. Datatype assigned this item in the Set Item Attributes option in Screen Builder.
<hr/>	
Name:	APPLICATION DISPLAY WINDOW
Call Sequence:	APPDIS(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be displayed. This routine must display all aspects of the window except the window border.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to display.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table G-2 Application-Provided Routines (Continued)

Name:	APPLICATION RECEIVE WINDOW
Call Sequence:	APPRCV(SCREEN, WINDOW, ITEM, KEY, UDWNBR, XPOS, YPOS)
Description:	This routine is provided by the developer. Window Manager calls it when information must be received from a user-defined window. This routine must handle all user input for the given window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window from which to receive information.
ITEM:	Input and Output. The input integer specifies the item upon which the cursor should be initially placed. The output integer specifies the item selected.
KEY:	Output. Integer code for the key pressed by the user.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).
XPOS:	Output. This is the integer indicating the X character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.
YPOS:	Output. This is the integer indicating the Y character coordinate of the cursor. It is used only if the KEY parameter returned is a window movement key code.

Name:	APPLICATION DELETE WINDOW
Call Sequence:	APPDEL(SCREEN, WINDOW, UDWNBR)
Description:	This routine is provided by the developer. Window Manager calls it when a user-defined window needs to be deleted. This routine must delete all nontext portions of the window.
Parameters:	
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen.
WINDOW:	Input. Window number of the user-defined window to be deleted.
UDWNBR:	Input. User-defined window code assigned to this window in the Set Window Format option in Screen Builder (the window type).

Table G-2 Application-Provided Routines (Continued)

Name:	APPLICATION DISPLAY MESSAGE
Call Sequence:	APPMSG(MSGSCR, MSGWND, KEY, MSGNBR, MSGTYP, SCREEN, WINDOW, ITEM)
Description:	This routine is provided by the developer. It is called when a user-defined message needs to be displayed. This routine must handle the display of the message, response from the user to the message, and the deletion of the nontext portion of the message from the screen.
Parameters:	
MSGSCR:	Input. Integer assigned by Window Manager to identify the screen containing the message window.
MSGWND:	Input. Window number of the user-defined message window.
KEY:	Output. Integer code for the key pressed by the user.
MSGNBR:	Input. Number of the message to be displayed. This is the number assigned to the message by the Message Builder utility.
MSGTYP:	Input. Integer indicating the type of message being displayed. The integer is one of the following values: 5 — Help 6 — Error 7 — Warning 8 — Please Note
SCREEN:	Input. Integer assigned by WMLOAD to identify the screen containing the window the cursor was in when the Help key was pressed. It is used only for Help messages.
WINDOW:	Input. Window number of the window the cursor was in when the Help key was pressed. It is used only for Help messages.
ITEM:	Input. Input integer specifies the item the cursor was on when the Help key was pressed. Value of -1 is passed when window level help is being displayed. It is used only for Help messages.

Name:	APPLICATION INPUT ROUTINE
Call Sequence:	APPINP()
Description:	This routine is provided by the developer. Window Manager calls it when input from the user is needed. It can be used to provide an alternative input method to the keyboard. This routine, if used, must still return valid key codes. A value of 0 must be returned if there is no key code ready to be returned.
Parameters:	None.

BASIC Compiling Requirements

G.5 Window Manager requires the use of Include files. These are provided on the Window Manager Run-Time Object diskette.

Include Files

G.5.1 An Include file containing attribute names for the WMSETV, WMGETV, WMSETS, and WMGETS Window Manager routines is provided to keep hard-coded constants out of the Compiled BASIC code. These constants are likely to change or at least expand with future releases of Window Manager. The Include filename is WMFIELD.BAS and can be included with the following statement:

```
REM $INCLUDE: 'WMFIELD.BAS'
```

The BASIC compiler checks all REM statements for these metacommands. Remember that the \$INCLUDE metacommand must be the last metacommand in a REM statement if other metacommands are used. In any programs that call the Window Manager routines to use window attributes (for example, WMGETV, WMSETS, and so on), you must include WMFIELD.BAS. If you include the WMFIELD.BAS file, you must compile the program with the =N option.

CAUTION: External declarations of routines are not required by the BASIC compiler. You must be very careful that the calls to the Window Manager routines are coded correctly with the proper number of variables, and that each is in the correct position. This is not checked as it is for other high-level languages. Be sure to consult Table G-1 for the calling sequences.

Table G-3 Data Declarations for the Compiled BASIC Interface

<i>Name</i>	<i>Type</i>	<i>Value</i>	<i>Description for Nonzero or True Condition</i>
REM Window Coordinates:			
LIwposux	=	1	' Left-most column
LIwposuy	=	2	' Top row
LIwposlx	=	3	' Right-most column
LIwposly	=	4	' Bottom row
REM Window format:			
LIawdtyp	=	5	' Window type
LIwincol	=	38	' Number of columns
LIinwprio	=	10	' Priority (for application use only)
LIinwmusl	=	11	' Multiple selection window
LIinwpopu	=	12	' Pop-up window
LIinpaint	=	46	' Special repaint on receive
LIwofset	=	21	' First item to be displayed
LIiautsc	=	40	' Show last item when painted
LIicentr	=	45	' Center all items
LIinorep	=	43	' Don't redisplay current items
LIinumul	=	42	' Disable multiple line items
LIcursin	=	74	' Allow cursor to enter window
LIidirct	=	44	' Multiple column order
LIilmlen	=	41	' Maximum item label length
LIiljust	=	47	' Item label justification
LIwshowf	=	20	' Pathname of window file
LIinwactv	=	13	' Window is active (read only access)
LIwitmb	=	39	' Number of items (read only access)
REM Active window attributes:			
LIablank	=	6	' Blinking
LIaunders	=	7	' Underline
LIarever	=	8	' Reverse video
LIactint	=	9	' Intensity/color
REM Inactive window attributes:			
LIinwblnk	=	16	' Blinking
LIinwundr	=	17	' Underline
LIinwrevr	=	18	' Reverse video
LIinwintn	=	19	' Intensity/color
REM Window Label attributes:			
LIwlabel	=	22	' Window label
LIwlvsbl	=	23	' Invisible label
LIwlpos	=	24	' Label position
LIwlcent	=	25	' Centered label
LIwlblnk	=	27	' Blinking label
LIwlundr	=	28	' Underlined label
LIwlrevr	=	29	' Reverse video label
LIwlint	=	30	' Intensity/color
LIwlaint	=	26	' Use item intensity

Table G-3 Data Declarations for the Compiled BASIC Interface (Continued)

<i>Name</i>	<i>Type</i>	<i>Value</i>	<i>Description for Nonzero or True Condition</i>
REM Cursor Attributes:			
LIcrsize	=	31	' Cursor size
LIcblink	=	33	' Blinking cursor
LIcunder	=	34	' Underlined cursor
LIcrever	=	35	' Reverse video cursor
LIcurint	=	37	' Cursor intensity/color
LIcalint	=	32	' Use Item intensity
LIcstays	=	36	' Cursor remains on a receive abort
REM Border Attributes:			
LIwbord	=	14	' Bordered window
LIbdrtsp	=	72	' Border takes up space (0 = Yes)
LIwscmk	=	15	' Display scroll markers
LIbdrrv	=	73	' Reverse video border
LIbdrint	=	74	' Border intensity/color
REM Item attributes:			
LIittext	=	66	' Item text
LIitchos	=	56	' Chosen/enable attributes
LIitvsbl	=	63	' Visible item
LIitunsl	=	59	' Unselectable item
LIitdspl	=	55	' Displayed
LIitmlen	=	48	' Maximum edit field length
LIitreqr	=	57	' Required edit field
LIitecho	=	58	' Echo edit field input
LIidatyp	=	65	' Edit field datatype
REM Item chosen active attributes			
LIitblnk	=	60	' Blinking chosen item
LIitundr	=	61	' Underlined chosen item
LIitrevr	=	62	' Reverse video chosen item
LIitint	=	64	' Chosen item intensity/color
REM Item label attributes:			
LIitlabl	=	54	' Item label
LIilblnk	=	50	' Blinking label
LIilundr	=	51	' Underlined label
LIilrevr	=	52	' Reverse video label
LIilint	=	53	' Label intensity/color
LIilaint	=	49	' Use item intensity
REM Other miscellaneous attributes:			
LIwmpath	=	71	' Window Manager help/phrase file path
LIpctype	=	75	' Machine type (read only access)
LIlcdon	=	76	' PRO-LITE lcd screen indicator

Memory Considerations

G.6 Because the NaturalLink Window Manager and the Compiled BASIC application program coexist in memory, sufficient memory to accommodate the code and data areas of each is required.

To achieve correct ordering of the code and data areas for the Compiled BASIC application program and Window Manager object, a routine has been added called LIMSBSH that defines the memory organization. This routine places the application program code in low memory, followed by the assembly language routines called by Window Manager, followed by the Window Manager code and the Window Manager data area. The application program's data area is in the same segment as the code for a Compiled BASIC-compiled program. The routine also declares the Window Manager stack and heap size, which must be statically allocated.

Get-memory errors appear when the Window Manager heap management routines run out of memory. The only way to resolve these errors, if they occur, is to increase the size of the Window Manager heap area or to reduce the number of screens (or screen sizes) that were loaded into memory at the time the error occurred.

The default stack and heap sizes for Window Manager are as follows:

2K stack = hexadecimal 800 bytes
32K heap = hexadecimal 8000 bytes

The maximum size of the Window Manager stack and heap is the size in bytes determined by the difference of hexadecimal FFF0 and the size of the Window Manager data segment (called NL_DATA and found in the application program's link map). It is recommended that at least hexadecimal 800 bytes be allocated for the Window Manager stack size. Using this minimum recommendation for the stack size, the maximum amount of memory that can be allocated for the heap area is hexadecimal F7F0 minus the size of the data segment NL_DATA. This will be approximately hexadecimal ED00 bytes for application programs that use only the Window Manager object and slightly less for those that also use Natural Language object. A change in the Window Manager stack and heap sizes will require changing the LIMSBSH.ASM source file, reassembling the routine, and relinking the application.

Linking Considerations

G.7 An example of a link control file used to link a simple Compiled BASIC application program is as follows. Most of the NaturalLink run time has been placed in libraries. The order of the libraries is significant, and they must be placed in the order shown.

LIMBSH+	Memory organization module. This must occur first.
mymain+	The BASIC application's main program.
...	Any application subroutines, if used.
demo /M	The .EXE file.
demo.MAP	The .MAP file.
mylib+	Any application libraries, if used.
LIMSB+	Library of language interface routines to Window Manager for Compiled BASIC.
WM+	Window Manager object library.
APPOTH+	Dummy Window Manager called application routines.
BASCOMG	BASIC run-time library (or BASRUNG).

Dummy Routines Library

G.7.1 The APPOTH library contains dummy routines for all application routines which are called by Window Manager. These include APPVAL, APPRCV, APPDIS, APPDEL, APPMSG, and APPINP. If you do not use the features provided by one or more of these routines, the reference to them will be resolved by the dummy modules in this library. If you have your own versions of these routines, they must be explicitly linked; if they are in a library, the library must come before APPOTH in the link stream. If you create any libraries of your application routines, these libraries must come before any NaturalLink libraries in the link stream.

WMKEYDEF.OBJ and WMSTRDEF.OBJ Files

G.7.2 The key definition file (WMKEYDEF.OBJ) and the internal phrase file (WMSTRDEF.OBJ) are included as part of the Window Manager library (WM.LIB). If you have created your own version of these files (by using the KBUILD utility for WMKEYDEF or by reassembling the source file for WMSTRDEF), you must explicitly link your own version. Your version of the files should be placed in the link stream following any application subroutines that you may have. The linker will then use your versions instead of the files in the library.

Object Code Segments

G.7.3 The object code for Window Manager is divided into two groups. Segments with the name NL_PROG contain executable code, while segments with the name NL_DATA contain static data, heap, and Window Manager run-time stack.

Restrictions

G.8 The maximum number of screens loaded from files is set to 20. A screen can have up to 255 windows and each window can have up to 65,536 items. Since screens are stored in the Window Manager heap, a screen that contains excessive windows and items can cause a get-memory error when loaded.

WINDOW MANAGER ERROR CODES

H

The following is a list of the error codes returned to the application from Window Manager and the high-level language interfaces. The codes are listed by number, followed by the error name and type. Note that the warning codes are negative numbers and error codes are positive numbers. The @1 symbol in an error message indicates where variable text, such as a file name, is to be inserted.

- 1 - Get memory error type: ERROR
The program was unable to get enough memory for its processing. When you press the **ENTER** key, you will be returned to the operating system. If several processes were run before receiving this error, reenter the program and run the process that reported the error first.
- 2 - Release memory error type: ERROR
The program was unable to release memory that it had allocated. If several processes were run before receiving this error, exit the program, reenter, and choose the process that reported the error first.
- 3 - Screen load error type: ERROR
An error was encountered while trying to load the @1 screen file. Exit and ensure that the screen file exists on the diskette and that it has not been damaged.
- 4 - Open file error type: ERROR
An error was encountered while trying to open the @1 file. If the file is new, check the availability of directory and disk space on your diskette. If the file is one that should exist on the diskette, check to be sure that it does exist and that the file has not been damaged.
- 5 - Close file error type: ERROR
An error was encountered while trying to close the @1 file. Check the directory and disk space on your diskette. Verify that your diskette has not been damaged.
- 6 - Read file error type: ERROR
An error was encountered while trying to read the @1 file. Check to be sure that the file exists on the diskette and that the diskette and file have not been damaged.
- 7 - Write file error type: ERROR
An error was encountered while trying to write to the @1 file. Check your available disk space to ensure that your diskette has not been damaged.

- 8 - Window not found type: WARNING
Window Manager could not find the window specified, so the request was ignored. This situation is probably the result of the window not being added. Windows must be added before Window Manager can do any processing.

- 9 - Window already exists type: WARNING
The window specified to be added has already been added. Window Manager will not add a window more than once, so the request was ignored.

- 10 - Window already active/inactive type: WARNING
The window to be selected or released is already active or inactive; the request was ignored.

- 11 - Window flagged to repaint type: WARNING
The window to be refreshed was already flagged to have its text and border repainted; the request was ignored.

- 12 - Receive from first active window type: WARNING
The Window Manager was unable to receive from the specified window and used the first valid, active window. A valid window is any active, nondisplay window that the cursor can enter, containing at least one selectable item.

- 13 - Invalid coordinates type: ERROR
The window specified has invalid position coordinates and cannot be added. Valid coordinate ranges are as follows:
 - Leftmost column 0 - 78
 - Top row 0 - 23
 - Rightmost column 1 - 79
 - Bottom row 1 - 24

- 14 - No valid windows type: ERROR
Window Manager could not find any valid windows from which to receive information. A valid window is any active, nondisplay window that the cursor can enter, containing at least one selectable item.

- 15 - Window type invalid type: ERROR
The window specified has an invalid window type and cannot be added. Valid window types are 0 through 4 and 10 through 99. Types 0 through 4 correspond to list, text, edit, file, and display windows, respectively. Types 10 through 99 are user-defined windows.

- 16 - No windows added type: ERROR
There have been no windows added yet. Windows must be added before Window Manager can do any processing.
- 19 - Cannot load file type: ERROR
The file text cannot fit into memory. The entire text of a file to be displayed in a file window must fit in memory before Window Manager will display the text. The window can still be displayed, but it will be blank and have no text associated with it.
- 20 - No file pathname type: ERROR
A file pathname was not supplied for the file window, so text was not placed in the window. The window can still be displayed, but it will be blank and will not have text associated with it.
- 21 - Empty show file type: ERROR
The file to be viewed contains no text and cannot be displayed. The window can still be displayed, but it will be blank and will not have text associated with it.
- 22 - Receive from inactive window type: WARNING
The Receive call is returning from an inactive window. Either the Proceed to Next Step key or an unknown key was pressed by the user when the cursor was in an inactive window.
- 23 - Window size problem type: ERROR
The window is not big enough. If a cursor is to be put in the window and the window has items defined, there must be room in the window to display at least one selectable item. A visible top or bottom window label occupies at least one line of the window. Check the top and bottom row coordinates of the window to ensure that they are a correct distance apart.
- 24 - Exit status from APPVAL type: WARNING
Window Manager receives a status of 2 (exit) from the application validation routine APPVAL.
- 29 - Not a screen type: ERROR
The @1 file is not a screen file. The file may have been damaged, the wrong file pathname may have been specified, or a different type of file may have been copied over the screen file desired.
- 51 - Items deleted type: WARNING
The item table just created has had items deleted from it.
- 52 - Fatal error type: ERROR
A fatal error has occurred. NaturalLink will no longer work, and subsequent calls to the interface will return with this error. This may be because NaturalLink could not access enough memory when copying strings from the application.

- 53 - Buffer error type: ERROR
The high-level language interface could not copy the string from NaturalLink or Window Manager back to the originating buffer because the buffer is not large enough. The string will be truncated at the length specified in the call.

- 54 - No more screens type: ERROR
There are no more entries available for additional screens. The screen cannot be added. The maximum number of screens that can be loaded at the same time is 20.

- 55 - Invalid screen number type: ERROR
The screen number specified was not a valid screen number. The screen number may be out of range.

- 56 - Unused screen number type: ERROR
The screen number specified does not have an active screen associated with it in memory at this time.

- 57 - Invalid window number type: ERROR
The window number specified in the call to Window Manager is invalid. The window number may be out of range.

- 58 - Invalid item number type: ERROR
The item number specified in the call to Window Manager is not a valid item number. The item number may be out of range.

- 59 - Invalid field name type: ERROR
The field name specified for getting or setting window attributes is invalid.

- 60 - Invalid field type type: ERROR
The type of field specified (string or integer) does not match the parameter type of the Window Manager routine that was called to return or set the value for the field.

- 72 - Invalid string type: ERROR
The string field specified or the length of the field is invalid.

a

- active window attributes *See* attributes, active window
- active windows *See* window
- Add Help Message to List command 4-19
- Add Item:
 - call 6-7
 - command 4-16
- Add Message command 5-5
- Add Window:
 - call 6-4
 - command 4-5
- algorithm, application validation
 - routine 7-7
- Allow Cursor to Enter Window
 - attribute 3-10
- APPDEL, delete call 8-10, D-15, E-15, F-16, G-20
- APPDIS, display call 8-10, D-14, E-14, F-15, G-19
- APPINP, application input routine 10-14
- application:
 - calls to Window Manager 8-11
 - Delete call 8-10
 - Display call 8-10
 - input routine 10-13
 - Message Manager call 8-11
 - program termination 6-10
 - Receive call 8-9
 - user-defined message calls 8-9
 - user-defined window calls 8-9
 - validation routine *See* validation routine, application
- application-defined function keys 2-5
- application input routine:
 - APPINP 10-14, D-16, E-16, F-17, G-21
 - dummy 10-14
- APPRCV, receive routine 8-9, D-15, E-15, F-15, G-20
- APPVAL, validation routine 7-5, D-14, E-14, F-15, G-19
- APP??? library D-21
- ASCII characters D-3, E-3, F-3, G-3
- assembly language routines G-15
- Attach Help Messages menu 4-18
- Attach Help to Item
 - command 4-16, 4-18
- Attach Help to Window
 - command 4-12, 4-18
- attributes, active window:
 - Blinking 3-12
 - Intensity/Color 3-12
 - Reverse Video 3-12
 - Underlining 3-12
- attributes, border:
 - Border Intensity Color 3-16
 - Border Takes Up No Space 3-15
 - Bordered Window 3-16
 - Display Scroll Markers 3-15
 - Reverse Video Border 3-15
- attributes, cursor:
 - Blinking Cursor 3-14
 - Cursor Intensity/Color 3-14
 - Cursor Size and Type 3-14
 - Don't Delete Cursor 3-15
 - Reverse Video Cursor 3-14
 - Underlined Cursor 3-14
 - Use Item Intensity 3-14
 - on IBM computers C-13
- attributes, inactive window:
 - Blinking 3-12
 - Intensity/Color 3-12
 - Reverse Video 3-12
 - Underlining 3-12
- attributes, item-chosen:
 - Blinking Chosen Item 3-19
 - Chosen Item Intensity/Color 3-19
 - Reverse Video Chosen Item 3-19
 - Underlined Chosen Item 3-19
- attributes, item format:
 - Chosen/Enable 3-19
 - Displayed 3-18
 - Echo Edit Field Input 3-18
 - Edit Field Datatype 3-18
 - Maximum Edit Field Length 3-18
 - Required Edit Field 3-18
 - Unselectable Item 3-17
 - Visible Item 3-17
- attributes, item label:
 - Blinking Label 3-13, 3-19
 - Label Intensity/Color 3-19
 - Reverse Video Label 3-19
 - Underlined Label 3-19
 - Use Item Intensity 3-19

- attributes, other 2-4
 - Allow Cursor to Enter Window 3-10
 - Blinking Chosen Item 3-19
 - Centered Items 3-10
 - Disable Multiple-Line Items 3-10
 - Don't Redisplay Current Items 3-10, 3-18, B-19
 - First Item to Be Displayed 3-9, 3-10, B-20
 - Item Label Justification 3-11
 - Item Label Length 3-11
 - Item-Level 3-11
 - Item Text 3-16
 - Multiple-Column Order 3-11
 - Multiple-Selection Window 3-8
 - Number of Items in Window 3-11
 - Numeric Value 3-16
 - Pop-Up Window 3-9
 - Show Last Item When Painted 3-9
 - Special Repaint on Receive 3-9, B-21, C-11
 - String Value 3-16
 - window 2-4, 3-3, 4-10
 - Window Label 3-3
 - Window Priority 3-8
 - Window-Type 3-6
- attributes, window label:
 - Blinking Label 3-13
 - Centered Label 3-13
 - Invisible Label 3-12
 - Label Intensity/Color 3-13, 3-19
 - Label Position 3-13
 - Reverse Video Label 3-13, 3-19
 - Underlined Label 3-13, 3-19
- automatic scrolling 3-9
- b**
 - Backspace function 10-10
 - Backup function 10-8
 - BASIC program language *See* MS-BASIC Compiled program language
 - blank lines B-16
 - Blinking Chosen Item attribute 3-19
 - Blinking Cursor attribute 3-14
 - Blinking Label attribute 3-13, 3-19
 - border attributes *See* attributes, border
 - borderless windows B-13
 - borders, shared 3-15
- c**
 - C program language *See* Lattice C program language
 - call:
 - Add Item 6-7
 - Add Window 6-4
 - application *See* application
 - Clear Screen 6-9
 - Create Item Table 6-8
 - Create Window 6-9
 - Delete Item 6-8
 - Delete Window 6-7
 - Display Message From Message Manager 6-10
 - Display Window 6-5
 - Get Attribute Value 6-9
 - Get String Value 6-9
 - Initialize Window Manager 6-3
 - Insert Item 6-8
 - Load Screen File 6-4
 - looping on Receive 3-15
 - Receive 3-9, 6-5
 - Receive From Window 6-5
 - Refresh Window 6-5
 - Release Window 6-6
 - Reset NaturalLink Memory 6-10
 - Reset Window Manager 6-7
 - Save Screen 6-7
 - Select Window 6-4
 - Set Attribute Value 6-9
 - Set String Value 6-9
 - Unload Screen 6-7
 - user-defined windows 8-4
 - Window Manager Receive 10-3
 - callable routines, Window Manager 6-3
 - callable routines:
 - Lattice C D-5
 - MS-BASIC Compiled G-5
 - MS-FORTRAN F-5
 - MS-Pascal E-5
 - calling sequence, typical 6-11
 - calls to Window Manager, application 8-11
 - carriage return characters B-5
 - Centered Items attribute 3-10
 - Centered Label attribute 3-13
 - Change Help File command 4-5, 4-16
 - Change Message Type/Class option 5-7
 - character strings, C program language D-3
 - characters:
 - fill 6-6
 - graphic values C-13
 - hexadecimal values C-13
 - checking, datatype 3-18, 7-3
 - Chosen/Enable attribute 3-8, 3-19, B-18
 - Chosen/Enable Item Intensity attribute C-11
 - Chosen Item Intensity/Color attribute, 3-19
 - Clear Screen call 6-9
 - code, user-defined window 8-3
 - columnar format 2-4, 3-7

- column-major items 3-11
 - columns, number of 3-7
 - command, Message Builder utility:
 - Add Message 5-5
 - Change Message Type/Class 5-7
 - Copy Message 5-7
 - Delete Message 5-8
 - List Messages 5-8
 - MBUILD 5-3
 - Modify Message 5-6
 - Quit 5-9
 - Rename Message 5-6
 - Reuse Message 5-7
 - Specify Window Coordinates 5-7
 - View Message 5-8
 - command, Phrase Builder utility:
 - Edit a Phrase 9-6
 - Exit 9-7
 - PBUILD 9-5
 - Print Phrases to File 9-6
 - Restore Default Phrases 9-6
 - Save Phrases to File 9-7
 - command, Screen Builder utility:
 - Add Help Message to List 4-19
 - Add Item 4-16
 - Add Window 4-5
 - Attach Help to Item 4-16, 4-18
 - Attach Help to Window 4-12, 4-18
 - Change Help File 4-5, 4-16
 - Copy Item 4-16
 - Copy Window 4-5
 - Delete Help Message From List 4-20
 - Delete Item 4-16
 - Delete Window 4-5
 - Draw Screen 4-6
 - Draw Window 4-13
 - Edit Item Label 4-15
 - Edit Item Table 4-13
 - Edit Item Text 4-14
 - Edit Window 4-4
 - Edit Window Label 4-11
 - Exit 4-8
 - Insert Help Message into List 4-20
 - Insert Item 4-16
 - Insert Window 4-5
 - List Screen Attributes 4-8
 - Load Screen 4-7
 - Save Screen 4-8
 - Set Active Window Attributes 4-10
 - Set Border Attributes 4-12
 - Set Cursor Attributes 4-12
 - Set Item Attributes 4-15
 - Set Item Label Attributes 4-16
 - Set Inactive Window Attributes 4-10
 - Set Window Format 4-10
 - Set Window Label Attributes 4-12
 - Set Window Position 4-10
 - Specify Help Message 4-20
 - Test Screen 4-7
 - Test Window 4-13
 - View Help Message 4-20
 - command, Set Function Keys utility 10-11
 - common blocks, MS-FORTRAN program
 - language F-18
 - compiler, Lattice C D-3
 - compiling requirements:
 - Lattice C D-17
 - MS-BASIC Compiled G-22
 - MS-FORTRAN F-18
 - MS-Pascal E-17
 - computer:
 - IBM C-3, C-12
 - Texas Instruments C-4
 - TI BUSINESS-PRO C-4
 - TI PRO-LITE C-5, C-9
 - TIPC C-5
 - TIPPC C-5
 - computer hardware Appendix C
 - coordinates, window 3-6
 - Copy Item command 4-16
 - Copy Message command 5-7
 - Copy Window command 4-5
 - Create Item Table call 6-8
 - Create Window call 6-9
 - cursor attributes *See* attributes, cursor
 - Cursor Intensity/Color attribute 3-14
 - cursor, invisible B-4
 - cursor movement 2-4, B-15, B-16, B-19
 - testing 4-7
 - user-defined windows 3-10, 8-4
 - Cursor Size and Type attribute 3-14
- d**
- datatype checking 3-18, 7-3
 - datatype parameter, application validation
 - routine 7-5
 - default:
 - function keys, how to change 10-11
 - keys 10-4, 10-5, 10-6, 10-7
 - message 5-9
 - definition, application validation
 - routine 7-3
 - Delete call 8-6, 8-10
 - Delete From the Cursor Out
 - function 10-10
 - Delete function 10-10
 - Delete Help Message From List
 - command 4-20

Delete Item:
 call 6-8
 command 4-16
Delete Message command 5-8
Delete Window:
 call 6-7
 command 4-5
detection, machine C-3
Disable Multiple-Line Items
 attribute 3-10
DISMSG Display Message call 6-10,
 D-13, E-13, F-13, G-14
display:
 call 8-5, 8-6
 user-defined messages 5-10
 window 2-3, B-4, B-15
Display Message From Message Manager
 call 6-10
Display Scroll Markers attribute 3-15
Display Window call 6-5
Displayed Item attribute 3-18, B-20
displays, multiple-window C-11
Don't Delete Cursor attribute 3-15
Don't Redisplay Current Items
 attribute 3-10, 3-18, B-19
Draw Screen command 4-6
Draw Window command 4-13
drawn screen printout 4-6
dummy:
 application input routine 10-14
 application validation routine 7-8
dummy routines library:
 Lattice C D-21
 MS-BASIC compiler G-26
 MS-FORTRAN F-22
 MS-Pascal E-20

e
Echo Edit Field Input attribute 3-18
Edit a Phrase command 9-6
Edit Field Datatype attribute 3-18
Edit Item Label command 4-15
Edit Item Table command 4-13
Edit Item Text command 4-14
edit window 2-3, B-4
Edit Window command 4-4
Edit Window Label command 4-11
edit windows, multiple-selection 7-3
editing keys 4-11, 5-5
environmental string, NLXTOOLS 4-3,
 5-3, 9-5, 10-11
equipment requirements Chapter 10
error:
 get memory 6-10
 message 1-7, 5-3

error codes:
 Message Manager utility 5-11
 Window Manager run-time Appendix H
Exit:
 command 4-8
 Phrase Editor command 9-7

f

features, Window Manager B-13
file:
 Help Message 4-16
 message 4-20, 5-3
 NLXPHRAS.NM\$ 9-7
 window 2-3, B-5
 window pathname 3-4
 window text files 3-7
 WMKEYDEF.OBJ 10-11, D-21, E-21,
 F-22, G-26
 WMSTRDEF.OBJ D-21, E-21, F-22, G-26
fill characters 6-6
First Item to Be Displayed attribute 3-9,
 3-10, B-20
free-format window 2-4, 3-8, 3-9, B-8
function:
 Backspace 10-10
 Backup 10-8
 Delete 10-10
 Delete From the Cursor Out 10-10
 Help 10-8
 Insert 10-10
 Move Left or Right One Item 10-9
 Move Left or Right One Word 10-9
 Move to Start or End of Line 10-9
 Next Active Window 10-9
 Next Item/Line Scrolling 10-8
 Next Window 10-9
 Page Scroll 10-7
 Printable Keys 10-10
 Printable Keys in Search Mode 10-11
 Proceed 10-7
 Select 10-7
 Top/Bottom 10-9
function key change utility 10-11, C-4
function keys:
 application-defined 2-5
 default, how to change 10-11
 differences C-5
 Window Manager 10-3

g

Get Attribute Value call 6-9
get memory error 6-10, D-19, E-19
 F-21, G-25
Get String Value call 6-9
graphic title windows 8-8

h

Help:

- attach 4-16
- file, setting a path for 4-17
- function 10-8
- key 2-5
- message file 4-16
- message, user-defined 4-20
- messages 1-7, 5-3, 8-11, 10-8
- messages, graphical 8-8
- helpful hints, Window Manager
Appendix B
- hexadecimal values of characters C-13

i

- IBM computers C-3, C-12
- icons 8-8
- inactive windows:
 - attributes *See* attributes,
inactive windows
 - on the PRO-LITE C-11
- Include file:
 - Lattice C D-17
 - MS-BASIC Compiled G-22
 - MS-FORTRAN F-18
 - MS-Pascal E-17
- initialize procedure 6-3
- input routine, application 10-13
- Insert function 10-10
- Insert Help Message into List
command 4-20
- Insert Item:
 - call 6-8
 - command 4-16
- Insert Window command 4-5
- integers:
 - Lattice C D-3
 - MS-BASIC Compiled G-3
 - MS-FORTRAN F-3
 - MS-Pascal E-3
- Intensity/Color attribute on IBM
computers C-13
- interface routines, MS-Compiled
BASIC G-5
- Internal Phrase Editing Chapter 9
- internal phrases:
 - definition 9-3
 - how to modify 9-4
- invisible:
 - cursor 3-14, B-4
 - item 3-17, B-13
 - label 3-12, B-6
 - Window item label 3-11, B-12

item:

- attributes 4-15
- column-major 3-11
- index field 6-6
- invisible 3-17, B-17
- label length 3-11
- labels 3-16, B-11
- multiple-column 3-7
- row-major 3-11
- truncation 3-11
- unselectable 3-17, B-16
- window 1-5
- item-chosen attributes *See* attributes, item-
chosen
- item format attributes *See* attributes, item
format
- item label attributes *See* attributes, item
label
- Item Label Justification attribute 3-11
- Item Label Length attribute 3-11
- item-level attributes 3-16
- Item-Level Commands menu 4-13
- Item Text attribute 3-16

k

- KBUILD 10-11; *See also* Set Function Keys
utility
- key:
 - codes C-5, C-7, C-13, C-16
 - default 10-4
 - editing 4-11
 - Help 2-5
 - printable 10-10, 10-11
 - Window Manager return 6-5
- keyboard input values, Window
Manager 10-4
- Keyboard layout:
 - BUSINESS-PRO C-4
 - IBM PC and PC/XT C-14
 - IBM Personal Computer AT C-14
 - PRO-LITE C-10
 - TIPC/TIPPC C-9
- keys used, application validation
routine 7-4

l

- label attributes *See* attributes, item label
- label:
 - invisible 3-12, B-6
 - item 3-16, B-11
 - window 1-5, 3-3, B-6
- Label Intensity/Color attribute
(window) 3-13, 3-19

Label Position attribute (window) 3-13

languages:

- Lattice C Appendix D
- MS-BASIC Compiled Appendix G
- MS-FORTRAN Appendix F
- MS-Pascal Appendix E

Lattice C program language:

- application routines D-14
- callable routines D-5
- character strings D-3
- compiler D-3
- compiling requirements D-17
- dummy routines library D-21
- include file D-17
- integers D-3
- linking considerations D-20
- memory considerations D-19
- memory model differences D-21
- parameters D-3
- restrictions D-22
- zero-base values D-3

LCD Flag, LlLCDon/lcdon C-12

line scrolling 2-4

linking considerations:

- Lattice C D-20
- MS-BASIC Compiled G-26
- MS-FORTRAN F-22
- MS-Pascal E-20

List Messages command 5-8

List Screen Attributes command 4-8

list window 2-3, B-3

LWmpath, string variable 4-17, 9-7

Load Screen command 4-7

Load Screen File call 6-4

looping on Receive call 3-15

m

machine detection Llptype/pctype
flag C-3

Maximum Edit Field Length
attribute 3-18

MBUILD command 5-3; *See also* Message
Builder utility

memory considerations:

- Lattice C D-19
- MS-BASIC Compiled G-25
- MS-FORTRAN F-21
- MS-Pascal E-19

memory model differences, C program
language D-21

menu:

- Attach Help Messages 4-18
- Item-Level Commands 4-13
- Screen Builder Utility 4-4

- Screen Level Commands 4-4
- Set Function Keys Utility 10-11
- Window Level Commands 4-9

message:

- default 5-9
- editing keys 5-5
- Error 1-7, 5-3
- file 4-20
- file pathname 5-3
- Help 1-7, 5-3, 8-11, 10-8
- Help, graphical 8-8
- options 5-3
- Please Note 5-3
- text 5-3
- type 5-5
- types 1-7
- user-defined 5-3, 5-9, 8-7, 8-8
- variable text 5-10
- Warning 1-7, 5-3

message application calls,
user-defined 8-9

Message Builder utility 1-7, 4-18, 4-20,
Chapter 5

Message Manager utility 1-7, 5-9, 8-11
call, application 8-11

error codes and messages 5-11

Modify Message command 5-6

Move Left or Right One Item
function 10-9

Move Left or Right One Word
function 10-9

Move to Start or End of Line
function 10-9

MS-BASIC Compiled program language:

- application-provided routines G-18
- callable routines G-5
- character strings G-3
- compiling requirements G-22
- dummy routines library G-26
- Include files G-22
- integers G-3
- interface routines G-5
- linking considerations G-26
- memory considerations G-25
- parameters G-3
- restrictions G-26
- zero-base values G-3

MS-DOS A-3

MS-DOS Link Editor A-3

MS-FORTRAN program language A-3
application-provided routines F-14

- callable routines F-5
- character strings F-3
- common blocks F-18

- compiling requirements F-18
- dummy routines library F-22
- Include files F-18
- integers F-3
- linking considerations F-22
- memory considerations F-21
- one-base values F-3
- parameters F-3
- restrictions F-23
- MS-Pascal program language A-3
 - application-provided routines E-14
 - callable routines E-5
 - character strings E-3
 - compiling requirements E-17
 - dummy routines library E-20
 - Include files E-17
 - integers E-3
 - linking considerations E-20
 - memory considerations E-19
 - parameters E-3
 - restrictions E-21
 - zero-base values E-3
- multiple-column:
 - items 3-7
 - window format B-9
- Multiple-Column Order attribute 3-11
- multiple-selection window B-17
 - edit windows 7-3
 - simulating B-18
- Multiple-Selection Window
 - attribute 3-8
- multiple-window:
 - displays C-11
 - effects B-13
- multi-purpose windows B-21

n

- NaturalLink menus 1-3
- Next Active Window function 10-9
- Next Item/Line Scrolling function 10-8
- Next Window function 10-9
- NLXPHRAS.NM\$ file 9-7
- NLXTOOLS 4-3, 5-3, 9-5
- Number of Items in Window
 - attribute 3-11
- Numeric Value attributes 3-16

o

- offset value 3-9
- one-base values, MS-FORTRAN program
 - language F-3

p

- Page Scroll function 10-7
- page scrolling 2-4
- parameters:
 - Lattice C D-3
 - MS-BASIC Compiled G-3
 - MS-FORTRAN F-3
 - MS-Pascal E-3
- Pascal *See* MS-Pascal program language
- pathname:
 - message file 5-3
 - phrase file 9-7
 - window file 3-4
- PBUILD command 9-5; *See also* Phrase Editor utility
- Phrase Editor utility 9-4
 - Edit a Phrase 9-6
 - Exit 9-7
 - Print Phrases to File 9-6
 - Restore Default Phrases 9-6
 - Save Phrases to File 9-7
- Please Note messages 1-7, 5-3
- Pop-Up Window attribute 3-9
- Print Phrases to File command 9-6
- printable keys 10-10, 10-11
- Printable Keys function 10-10
- Printable Keys in Search Mode
 - function 10-11
- procedure calls, Window Manager 1-8
- Proceed function 10-7
- pseudo code, Window Manager Receive call 8-6

q

- Quit command 5-9

r

- Receive calls 3-9, 6-5
 - application 8-9
 - looping on 3-15
 - pseudo code, Window Manager 8-6
 - user-defined windows 8-4
 - Window Manager 10-3
- Receive From Window call 6-5
- Refresh Window call 6-5
- Release Window call 6-6
- Rename Message command 5-6
- Required Edit Field attribute 3-18
- Reset NaturalLink Memory call 6-10
- Reset Window Manager call 6-7
- Restore Default Phrases Command 9-6

- return status code
 - application validation routine 7-5
 - Window Manager callable routines D-5, E-5, F-5, G-5
- Reuse Message command 5-7
- Reverse Video attribute 3-12, C-10, C-13
- Reverse Video Border attribute 3-15
- Reverse Video Chosen Item attribute 3-19
- Reverse Video Cursor attribute 3-14
- Reverse Video Window Label attribute 3-13
- routines:
 - APPINP (application input) 10-14
 - application input 10-13
 - assembly language G-15
 - dummy application input 10-14
 - Window Manager callable 6-3
- row-major items 3-11
- S**
 - Save Phrases to File command 9-7
 - Save Screen:
 - call 6-7
 - command 4-8
 - SBUILD command 4.1; *See also* Screen Builder utility
 - screen 1-6
 - file design B-22
 - Screen Builder utility 1-7, Chapter 4; *See also* commands, Screen Builder menu 4-4
 - Screen-Level Commands menu 4-4
 - scrolling:
 - automatic 3-9
 - line 2-4
 - one item at a time 2-4
 - one window at a time 2-4
 - page 2-4
 - search 2-4
 - Select function 10-7
 - Select Window call 6-4
 - Set Active Window Attributes command 4-10
 - Set Attribute Value call 6-9
 - Set Border Attributes command 4-12
 - Set Cursor Attributes command 4-12
 - Set Function Keys utility 10-11
 - Set Item Attributes command 4-15
 - Set Item Label Attributes command 4-16
 - Set Inactive Window Attributes command 4-10
 - Set String Value call 6-9
 - Set Window Format command 4-10

- Set Window Label Attributes command 4-12
- Set Window Position command 4-10
- shared borders 3-15
- Show Last Item When Painted attribute 3-9
- single-column window format B-9
- Special Repaint on Receive attribute 3-9, B-21, C-11
- Specify Help Message command 4-20
- Specify Window Coordinates command 5-7
- String Value attribute 3-16

t

- Test Screen command 4-7
- Test Window command 4-13
- Texas Instruments computers C-4
- text:
 - message 5-3
 - window 2-3, B-4
 - window files 3-7
- TI BUSINESS-PRO computer C-4
- TI PRO-LITE computer C-5, C-9
- TIPC computer C-5
- TIPPC computer C-5
- Top/Bottom function 10-9
- truncation 3-10, 3-11, 3-16
- typical calling sequence 6-11

u

- UDM *See* user-defined message
- UDW *See* user-defined window
- Underlined Chosen Item attribute 3-19
- Underlined Cursor attribute 3-14
- Underlined Label attribute 3-13, 3-19
- Underlining attributes
 - active windows 3-12
 - inactive windows 3-12
 - on IBM computers C-12
- Unload Screen call 6-7
- Unselectable Item attribute 3-17
- unselectable items 3-17, B-16
- Use Item Intensity attribute 3-14
- user-defined message 5-3, 5-5, 8-7
 - application calls 8-9
 - display 5-10
 - Help message 4-20
 - uses 8-8
 - view 5-8
- user-defined windows:
 - application calls 8-9
 - code 8-3

- control by application 8-3
- control by Window Manager 8-4
- cursor movement 8-4
- Delete call 8-6
- Display call 8-6
- how to specify 8-3
- processing help 8-4
- Receive call 8-4
- uses 8-8
- utility:
 - Message Builder 1-7, 4-18, 4-20, Chapter 5
 - Message Manager 1-7, 5-9, 8-11
 - Phrase Editor 9-4
 - Screen Builder 1-7, 4-3, Chapter 4
 - Screen Builder menu 4-4
 - Set Function Keys 10-11

V

- validation routine, application:
 - algorithm 7-7
 - application Chapter 7
 - datatype parameter 7-5
 - definition 7-3
 - dummy application 7-8
 - item number parameter 7-6
 - keys used 7-4
 - parameters 7-5
 - restrictions 7-8
 - return status code 7-5
 - steps 7-4
 - when to use 7-3
- value, offset 3-9
- variable, string 9-7
- Variable Text messages 5-10
- View Help Message command 4-20
- View Message command 5-8
- Visible Item attribute 3-17

W

- Warning messages 1-7, 5-3
- window:
 - active 1-6, 3-5, 3-10, 3-11, 3-19
 - borderless B-13
 - code, user-defined 8-3
 - columnar-format 3-7, B-8
 - coordinates 3-6
 - description 1-5, 3-3
 - display 2-3, B-4, B-15
 - edit 2-3, B-4
 - file 2-3, B-5
 - free-format 3-8, 3-9, B-8
 - graphic title 8-8
 - inactive on PRO-LITE computer C-11

- item 1-5
- label 1-5, 3-3, B-6
- list 2-3, B-3
- multi-purpose B-21
- multiple-selection B-17
- pathname, file 3-4
- size 3-6
- text 2-3, B-4
- types 2-3, 3-6, B-3
- user-defined 2-4, Chapter 8
- window attributes *See* attributes, other
- window format 2-4, B-8
 - attributes 3-6
 - free-format B-10
 - multiple-column B-9
 - single-column B-9
- window label attributes 3-3, 3-12
- Window Level Commands menu 4-9
- Window Manager:
 - callable routines 6-3
 - description of 1-5
 - features Chapter 2, B-13
 - function keys 10-3
 - helpful hints B-3
 - keyboard input 10-4
 - procedure calls 1-8, 6-3
 - Receive call 10-3
 - Receive call pseudo code 8-6
 - return key 6-5
 - routines, application-provided D-14, E-14, F-14, G-15
 - strings D-3, E-3, F-3, G-3
- window position attributes 3-6
- Window Priority attribute 3-8
- WMCLRS 6-9, D-13, E-13, F-13, G-14
- WMFLSH 6-10, D-13, E-13, F-13, G-14
- WMGETS 6-9, D-11, E-11, F-11, G-12
- WMGETV 6-9, D-11, E-10, F-11, G-12
- WMIADD 6-7, D-9, E-9, F-9, G-10
- WMICRE 6-8, D-10, E-10, F-10, G-11
- WMIDEL 6-8, D-10, E-9, F-10, G-10
- WMIINS 6-8, D-9, E-9, F-9, G-10
- WMINIT 6-3, D-6, E-6, F-6, G-6
- WMKEYDEF.OBJ file 10-11, D-21, E-21, F-22, G-26
- WMRSET 6-7, D-6, E-6, F-6, G-6
- WMSETS 6-9, D-12, E-12, F-12, G-13
- WMSETV 6-9, D-12, E-11, F-12, G-13
- WMSTRDEF.OBJ file D-21, E-21, F-22, G-26
- WMUNLD 6-7, D-9, E-9, F-9, G-10
- WMWADD 6-4, D-6, E-6, F-6, G-7
- WMWCRE 6-9, D-10, E-10, F-10, G-11
- WMWDEL 6-7, D-8, E-8, F-8, G-9
- WMWDIS 6-5, D-7, E-7, F-7, G-8

WMWRCV 6-5, D-8, E-7, F-8, G-8
WMWREF 6-5, D-7, E-7, F-7, G-8
WMWREL 6-6, D-8, E-8, F-8, G-9
WMWSAV 6-7, D-9, E-8, F-9, G-9
WMWSEL 6-4, D-7, E-7, F-7, G-7

Z

zero-base values:

Lattice C D-3
MS-BASIC G-3
MS-Pascal E-3

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

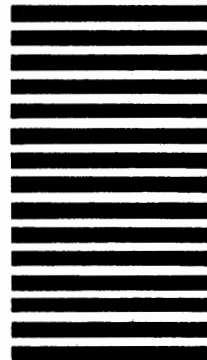
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DATA SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769



FOLD



TEXAS
INSTRUMENTS

