SIBO 'C' Software Development Kit

HWIM REFERENCE

Version 2.11

February 3, 1995

(C) Copyright Psion PLC 1990-95

All rights reserved. This manual and the programs referred to herein are copyrighted works of Psion PLC, London, England. Reproduction in whole or in part, including utilization in machines capable of reproduction or retrieval, without express written permission of Psion PLC, is prohibited. Reverse engineering is also prohibited.

The information in this document is subject to change without notice.

Psion and the Psion logo are registered trademarks, and Psion, PsionMC, Psion HC, Psion Series 3, Psion Series 3a and Psion Work*about* are trademarks of Psion PLC.

TopSpeed is a registered trademark of Clarion Software Corporation. Intel 8086 and 80286 are registered trademarks of Intel Corporation. IBM, IBM XT and IBM AT are registered trademarks of International Business Machines Corp. Microsoft and MS-DOS are registered trademarks of Microsoft Corporation. Apple and Macintosh are registered trademarks of Apple Computer Inc. VAX and VMS are registered trademarks of Digital Equipment Corporation. Brief is a registered trademark of Underware Inc. Psion PLC acknowledges that some other names referred to are registered trademarks.

CONTENTS

1 Introduction 1-1
Using HWIM classes 1-1

Notation 1-2
Names 1-2
Method function prototypes 1-2
The 'no leave' symbol 1-2
Class diagrams 1-3
Class hierarchy 1-3

Structured Error Recovery 1-4
Use of the p_leave mechanism 1-4
Panic numbers 1-4

2 The HWIMMAN Application Manager 2-1
Precursors 2-1
Class diagram 2-2
Class definition 2-2
Property 2-3

HWIMMAN methods 2-3
Initialise 2-3
Generate resource file name 2-5
Wait for an event 2-6
Clean up resources and report an error 2-6
Record a new filename 2-7
Report an error 2-7
Wait for all activity to cease 2-7
Find application image file 2-7
Guarantee existence of IPCS 2-8

3 The WSERV Class 3-1
Precursors 3-2
Class diagram 3-2
Class definition 3-2
Property 3-6

WSERV methods 3-9
Application initialisation 3-9
Initialise 3-9
Queue a message from the server 3-10
Cancel 3-10
Cancel any pending window server read 3-10
Process a message from the server 3-11
Log a new client window 3-14
Run a dialog 3-15
Add a dialog to the dialog list 3-15
Remove a dialog from the dialog list 3-16
Paragraph word wrap 3-16
Run help system 3-16
Get choice list resource text 3-16

Run the free-form dialling dialog ... 3-17
Process a Switchfiles message ... 3-17
Alter the lock count ... 3-17
Set alternative menu bar ... 3-17
Reset the menu bar ... 3-17
Run a submenu ... 3-18
Run a query dialog ... 3-18
Run an error dialog ... 3-19
Evaluate an expression ... 3-19
Set or get evaluator environment variable ... 3-19
Set or get dial environment variable ... 3-20
Run the set format dialog ... 3-20
Interface to wsAlert ... 3-21
Run country selector dialog ... 3-21
Smart dial of a number ... 3-21
Ensure print context data exists ... 3-22
Run print setup dialog ... 3-22
Run printer configuration dialog ... 3-22
Sense text for current printer device ... 3-22
Add a filelist to the list ... 3-22
Remove a filelist from the list ... 3-23
Animator has ticked over ... 3-23
Unrecognised WSERV event ... 3-23
Foreground message ... 3-23
Background message ... 3-23
Request date change notification ... 3-23
Process WM_DATE_CHANGED ... 3-24
Launch a DYL ... 3-24
Set up a status window 'diamond' list ... 3-24
Display allocator statistics ... 3-25
Copy print context from a process ... 3-25
Perform internal data check ... 3-26
Hide application from System Screen ... 3-26
Set attached state ... 3-26
Display a file-related message ... 3-27
Run Agenda memo editor ... 3-27
Run dialog in another process ... 3-27
Exit application with alert ... 3-28

SHUTTER ... 3-28
Processing a Switchfiles message ... 3-28
Class definition ... 3-29
Property ... 3-29

SHUTTER methods ... 3-29
Initialise ... 3-29
Do, or prepare for, the file switch ... 3-30
Handle error ... 3-30

4 The Command Manager ... 4-1
Precursors ... 4-1
Class diagram ... 4-2

COMMAN ... 4-2
Class definition ... 4-2
Property ... 4-2

COMMAN methods ... 4-3
Initialise ... 4-3
Toggle permanent status window ... 4-3
Validate a command ... 4-3
A pull-down menu is about to appear ... 4-3
Process a W_KEY_MODE key ... 4-4
Open or create a file ... 4-4
Exit the application ... 4-5

5 Windows **5-1**
Precursors 5-1
Class diagram 5-1

The WIN Class 5-2
Class definition 5-2
Property 5-4
Window flags 5-4

WIN methods 5-4
Destroy 5-4
Create the window's window server data 5-5
System-initiated redraw 5-5
Application-initiated redraw 5-5
Set visibility 5-6
Set window highlight 5-6
Sense start ID for help 5-6
Calculate a window position 5-6
Set window position 5-7
Process a keypress 5-7

Deferred WIN methods 5-7
Initialise 5-7
Set property 5-8
Sense property 5-8
Draw to existing GC 5-8

The BWIN bordered window class 5-9
Class definition 5-9
Property 5-9

BWIN methods 5-9
Draw border 5-9
Update border 5-10

The LODGER class 5-10
Class definition 5-11
Property 5-11

LODGER methods 5-12
Initialise 5-12
Destroy 5-12
Set visibility 5-12
Create GC and draw 5-12
Check content is valid 5-13

Deferred LODGER methods 5-13
Sense required width 5-13
Update file name 5-13

6 List Boxes and Menus **6-1**
Precursors 6-1

List boxes 6-2
Class definition 6-3
Property 6-4
List box flags 6-5

LISTBOX methods 6-6
Destroy 6-6
Initialise 6-6
Handle key input 6-7
Draw 6-8
Set emphasis 6-8
Set the window size and position 6-9
Draw an item 6-9
Toggle emphasis for an item 6-10
Sense required width for an item 6-10

Move focus to an item ... 6-10
Sense the item with focus ... 6-10
Get pointer to item text ... 6-10
Get index of last item ... 6-11

The Menu Bar ... 6-11
Class definition ... 6-12
Property ... 6-13

MENUBAR methods ... 6-13
Displaying a pull-down menu ... 6-13
Destroy ... 6-14
Initialise ... 6-14
Set visibility ... 6-14
Draw ... 6-15
Handle key input ... 6-15
Add a menu ... 6-16

PULLDOWN pull-down menu ... 6-16
Class definition ... 6-17
Property ... 6-17

PULLDOWN methods ... 6-17
Handle key input ... 6-17
Draw an item ... 6-17
Set item emphasis ... 6-18
Sense required width for an item ... 6-18

XPULLDWN pull-down menu ... 6-18
Class definition ... 6-19
Property ... 6-19

XPULLDWN methods ... 6-19
Connect to the window server ... 6-19
Draw ... 6-19
Set item emphasis ... 6-19

MENUTAB ... 6-20
Class definition ... 6-20
Property ... 6-20

MENUTAB methods ... 6-20
Initialise ... 6-20
Draw ... 6-20

7 Dialog Boxes ... 7-1
Precursors ... 7-1
Class diagram ... 7-1

DLGCHAIN ... 7-2
Class definition ... 7-2
Property ... 7-2
DLGCHAIN flags ... 7-2

DLGBOX ... 7-3
Class definition ... 7-4
Property ... 7-7
DLGBOX flags ... 7-8
Small font dialogs for the Workabout ... 7-9
DLGBOX_ITEM flags ... 7-9

DLGBOX methods ... 7-10
Consistency checks ... 7-10
Initialise ... 7-11
Destroy ... 7-11
Add an item ... 7-12
Add an item by resource ID ... 7-13
Replace an existing item ... 7-13

Set item by index 7-14
Change the prompt for an item 7-14
Sense item by index 7-14
Handle a keypress 7-15
Handle key input 7-16
Move focus to specified item 7-17
Draw whole dialog 7-17
Set size of dialog 7-18
Dim an item 7-19
Lock an item 7-19
Display 'dimmed' message 7-19
Sense item index 7-19
Sense item handle 7-20
Set item flags 7-20
Sense start ID for Help 7-20
Set dialog box emphasis 7-20
Specify minimum widths 7-20
Dynamic initialisation 7-21

Deferred DLGBOX methods 7-21
Item changed message 7-21
Focus changed message 7-21
Launch sub-dialog if required 7-21
Create non-system dialog item 7-22

ATSDIAL 7-22
Class definition 7-23
Property 7-23

ATSDIAL methods 7-24
Destroy 7-24
Add an item 7-24
Dynamic initialisation 7-24
Handle key input 7-25

8 Labels, Buttons and Choice Lists 8-1
Precursors 8-1
Class diagram 8-1

The TEXTWIN class 8-2
Class definition 8-3
Property 8-4
TEXTWIN flags 8-4

TEXTWIN methods 8-5
Initialise 8-5
Draw text 8-5
Set property 8-5
Sense text 8-6
Handle key input 8-6
Return required control width 8-7
Emphasise 8-7

The SMACLIST class 8-7
Class definition 8-9
Property 8-9

SMACLIST methods 8-10
Destroy 8-10
Initialise 8-10
Handle key input 8-11
Draw 8-11
Sense data 8-11
Set id, width and button positions 8-11
Sense required width 8-12

The ACLIST class ... 8-12
Class definition ... 8-14
Property ... 8-14

ACLIST methods ... 8-15
Initialise ... 8-15
Draw ... 8-15
Sense required width ... 8-15

The CHLIST class ... 8-16
Class definition ... 8-18
Property ... 8-18
CHLIST flags ... 8-19

CHLIST methods ... 8-19
Destroy ... 8-19
Initialise ... 8-19
Handle key input ... 8-19
Draw ... 8-20
Set current item and/or data ... 8-21
Sense data and selected item ... 8-21
Sense required width ... 8-21
Set emphasis ... 8-21

The NCHLIST numeric choice list class ... 8-22
Class definition ... 8-22
Property ... 8-22

NCHLIST methods ... 8-22
Set current item and/or maximum value ... 8-23
Sense selected item ... 8-23

The VARES resource array class ... 8-23
Class definition ... 8-24
Property ... 8-24

VARES methods ... 8-24
Copy a record ... 8-24
Get record length ... 8-24
Initialise ... 8-24
Delete sequence of records ... 8-24
Insert sequence of records ... 8-25
Point to record ... 8-25
Point to record data ... 8-25
Compress ... 8-25

The VANUMBER numeric array class ... 8-26
Class definition ... 8-26
Property ... 8-26

VANUMBER methods ... 8-26
Destroy ... 8-26
Point to record data ... 8-27
Set maximum value ... 8-27

9 Numeric Editors ... 9-1
Precursors ... 9-1
Class diagram ... 9-1

MFNE ... 9-2
Class definition ... 9-2
Property ... 9-3

MFNE methods ... 9-5
Handle key press ... 9-5
Draw current string ... 9-6
Set data ... 9-6
Emphasise ... 9-6

Return minimum width 9-6
Validate the current number 9-6
Show range information message 9-7
MFNE subclass examples 9-8
A basic subclass 9-8
Extended range date editor control 9-9
LNCEDIT 9-13
Class definition 9-14
Property 9-14
LNCEDIT methods 9-14
Initialise editor 9-14
Set data 9-15
Sense value 9-15
NCEDIT 9-16
Class definition 9-17
Property 9-17
NCEDIT methods 9-17
Initialise 9-17
Set new values 9-18
Sense new value 9-18
WNCEDIT 9-19
Class definition 9-20
Property 9-20
WNCEDIT methods 9-20
Initialise 9-20
Set new values 9-21
Sense new value 9-21
DTEDIT 9-22
Class definition 9-24
Property 9-25
DTEDIT methods 9-25
Initialise 9-25
Set new values 9-26
Sense value 9-26
Handle a keypress 9-27
Emphasise 9-27
Validate values and fields 9-27
LLEDIT 9-28
Class definition 9-29
Property 9-30
LLEDIT methods 9-30
Initialise 9-30
Set new values 9-30
Sense current value 9-31
RGEDIT 9-31
Class definition 9-32
Property 9-33
RGEDIT methods 9-33
Initialise 9-33
Set new values 9-33
Sense new value 9-33

10 Text Editors 10-1
Precursors 10-1
Class diagram 10-1

EDWIN 10-2
The edit window 10-2
Text lines 10-2
Cursor position 10-3
Line cursor 10-3
Link paste 10-3
Class definition 10-3
Property 10-5

EDWIN methods 10-6
Destroy 10-6
Create components and initialise 10-6
Handle key input 10-10
Draw view 10-15
Sense help ID 10-15
Set text 10-15
Sense text 10-15
Set emphasis 10-16
Set window ID and position 10-16
Sense width 10-16
Sense screen image size 10-16
Set screen image size data 10-16
Set content and cursor 10-17
Sense cursor and select data 10-18
Insert at cursor 10-19
Replace select and add surrounds 10-19
Set font and line spacing 10-19
Leave handling 10-20
Search for text and show cursor 10-20
Replace select 10-20
Evaluate an expression 10-21
Copy select to clipboard 10-22
Paste in clipboard contents 10-22
Perform bring 10-22
Insert text 10-23
Return keypress handling 10-23
Tab keypress handling 10-24
Customize style 10-24
Return read-only state 10-24

PUNCTUED 10-24
Class definition 10-25
Property 10-25

PUNCTUED methods 10-25
Handle key input 10-25

FLTEDIT 10-26
Class definition 10-26
Property 10-27

FLTEDIT methods 10-27
Initialise 10-27
Set values 10-28
Sense current value 10-28
Validate fields 10-29

XEDIT 10-29
Class definition 10-30
Property 10-30

XEDIT methods 10-30
Get required width 10-30
Handle key input 10-30
Draw 10-31
Sense data 10-31
Clear password data 10-31
Set emphasis 10-31

11 Gauge Classes **11-1**
Precursors 11-1
Class diagram 11-1

GAUGE 11-1
Class definition 11-2
Property 11-2

GAUGE methods 11-2
Initialise 11-2
Set characteristics of gauge sections 11-3
Return minimum width for gauge 11-3
Draw the gauge 11-3

DONEWN 11-3
Class definition 11-4
Property 11-5

DONEWN methods 11-5
Set gauge 11-5

Using Gauges 11-5
A GAUGE class example 11-5
A DONEWN class example 11-6
A dynamic gauge 11-7
Annotating a gauge 11-10

12 File Selectors **12-1**
Precursors 12-1

VADEV 12-1
Class diagram 12-2
Class definition 12-2
Property 12-2

VADEV methods 12-2
Initialise 12-2
Search for a device 12-2
Return public device specification 12-2

PACKSEL 12-3
Class diagram 12-4
Class definition 12-4
Property 12-4

PACKSEL methods 12-4
Initialise 12-4
Set pack from file name 12-5
Handle key input 12-5
Validate drive 12-5

FNEDIT 12-6
Class diagram 12-7
Class definition 12-7
Property 12-7

FNEDIT methods 12-8
Initialise 12-8
Set file name 12-9
Sense full file name 12-9
Handle key input 12-9
Emphasise 12-10
Validate file name 12-10
Update with new path 12-11

FNSELWN 12-11
Class diagram 12-12
Class definition 12-12
Property 12-13

Tagged files 12-13

FNSELWN methods 12-13
Destroy 12-13
Initialise 12-14
Change directory and/or selection 12-14
Get file name 12-16
Handle key input 12-16
Validate file name 12-17
Sense width 12-17
Update filename 12-17
Insert tags 12-18
Extract tags 12-18
Non-standard work on file list 12-18
Validate file list 12-18

13 File List Generator Classes 13-1
Precursors 13-1
Class diagram 13-1

NONODE 13-1
Class definition 13-2
Property 13-2

NONODE methods 13-2
Initialise 13-2

HPSEL 13-3
Class definition 13-3
Property 13-3

HPSEL methods 13-3
Initialise 13-3
Queue 13-4
Start scan 13-4
Data rebuilt 13-4

14 The FILELIST Window Class 14-1
Precursors 14-1

FILELIST 14-1
Class diagram 14-2
Class definition 14-3
Property 14-4

FILELIST methods 14-6
Destroy 14-6
Initialise file list 14-6
Handle key input 14-7
Return current selection 14-10
Draw an item 14-10
Get item text 14-11
List has been generated 14-12
Check for change in LOC:: 14-12

15 General System Dialogs 15-1
Precursors 15-1
Class diagram 15-2

ERRORDLG 15-2
Class definition 15-3
Resource 15-3
Property 15-4

ERRORDLG methods 15-4
Dynamic initialisation 15-4
QUERYDLG 15-4
Class definition 15-5
Property 15-5
Resource 15-5
QUERYDLG methods 15-6
Dynamic initialisation 15-6
Add item to end of list 15-6
FLISTDLG 15-6
Class definition 15-7
Property 15-7
Resources 15-7
FLISTDLG methods 15-7
Dynamic initialisation 15-7
Handle key input 15-8
FONTSEL 15-8
Class definition 15-9
Property 15-9
Resources 15-10
FONTSEL methods 15-11
Dynamic initialisation 15-11
Handle key input 15-11
Handle item changed messages 15-12
EVALDLG 15-12
Class definition 15-13
Property 15-13
Resource 15-14
EVALDLG methods 15-14
Dynamic initialisation 15-14
Handle key input 15-15
Handle item changed messages 15-16
Set minimum default sizes 15-16
SETPORTDLG 15-17
Class definition 15-17
Property 15-17
Resources 15-18
SETPORTDLG methods 15-20
Initialise dialog items 15-20
Handle key input 15-20
SETHSHKDLG 15-21
Class definition 15-21
Property 15-22
Resources 15-22
SETHSHKDLG methods 15-23
Initialise items 15-23
Handle key input 15-23

16 Print Classes 16-1
Precursors 16-3
Class diagram 16-3
LPRINTER 16-4
Print elements 16-4
Class definition 16-5
Property 16-6

LPRINTER methods 16-6
Destroy 16-6
Start printing dialog 16-7
Read call-back 16-7
Get printer text width 16-8

Deferred LPRINTER methods 16-8
Get text to print 16-8

PDEVDLG 16-9
Class definition 16-10
Property 16-11
Resources 16-11

PDEVDLG methods 16-12
Destroy 16-12
Launch sub-dialog 16-12
Dynamic initialisation 16-13
Handle key input 16-14
Read in string resources 16-14
Handle item changed messages 16-14

PRNPREV 16-15
Class definition 16-15
Property 16-15
Resources 16-15

PRNPREV methods 16-16
Dynamic initialisation 16-16
Handle key input 16-17

PRNCTRL 16-17
Class definition 16-19
Property 16-19
Resources 16-19

PRNCTRL methods 16-21
Dynamic initialisation 16-21
Launch sub-dialog 16-21
Change default minimum sizes 16-22
Handle key input 16-22

PAGECTRL 16-23
Class definition 16-24
Property 16-24
Resources 16-24

PAGECTRL methods 16-24
Dynamic initialisation 16-24
Handle key input 16-25

MARGINS 16-25
Class definition 16-26
Property 16-26
Resources 16-26

MARGINS methods 16-27
Dynamic initialisation 16-27
Handle key input 16-27

HEADFOOT 16-28
Class definition 16-29
Property 16-29
Resources 16-29

HEADFOOT methods 16-30
Dynamic initialisation 16-30
Handle key input 16-31
Launch sub-dialog 16-31

PRNMODEL 16-32
Class definition 16-32
Property 16-33
Resources 16-33

PRNMODEL methods 16-34
Dynamic initialisation 16-34
Handle key input 16-35
Launch sub-dialog 16-35
Handle item changed messages 16-35

PAGESIZE 16-36
Class definition 16-36
Resources 16-36

PAGESIZE methods 16-37
Dynamic initialisation 16-37
Handle key input 16-38
Handle item changed messages 16-38

PRINTING 16-39
Class definition 16-40
Property 16-40
Resources 16-41

PRINTING methods 16-41
Dynamic initialisation 16-41
Set size of dialog 16-42
Set title 16-42
Start print 16-42
Completion call-back 16-42

PMODLOCS 16-43
Class diagram 16-43
Class definition 16-44
Property 16-44

PMODLOCS methods 16-44
Process matching filename 16-44

17 Dialling Dialogs 17-1
Precursors 17-1

HAIDLE 17-1
Class diagram 17-2
Class definition 17-2
Property 17-2

HAIDLE methods 17-2

DIALAO 17-3
Class diagram 17-3
Class definition 17-3
Property 17-3

DIALAO methods 17-4
Start dialling object 17-4
Handle filter keys 17-4

DIALDLG 17-5
Class diagram 17-5
Class definition 17-5
Property 17-5

DIALDLG methods 17-6
Create a dial active object 17-6

FREEDIAL 17-6
Class diagram 17-6
Class definition 17-7

Property 17-7

FREEDIAL methods 17-7
Destroy 17-7
Initialise freeform dialling 17-7
Draw dial string 17-7
Add input and dial 17-8
Sense lodger width 17-8

FDIALDLG 17-9
Class diagram 17-9
Class definition 17-9
Property 17-9
Resources 17-10

FDIALDLG methods 17-10
Handle key 17-10

CNTRYDLG 17-11
Class diagram 17-12
Class definition 17-12
Property 17-12
Resources 17-12

CNTRYDLG methods 17-12
Handle key 17-12

SDIALDLG 17-13
Class diagram 17-13
Class definition 17-13
Property 17-14
Resources 17-14

SDIALDLG methods 17-15
Add one or more items to the dialog 17-15
Initialise dialog 17-16
Handle key 17-16

18 Help Classes 18-1
Precursors 18-3

HELPLIST 18-4
Class diagram 18-5
Class definition 18-5
Property 18-6

HELPLIST methods 18-6
Initialise help list 18-6
Handle keys 18-7
Get item text 18-7
Get width of item 18-7
Format and draw an item 18-8
Draw or remove emphasis on item 18-8

HELPDLG 18-8
Class diagram 18-9
Class definition 18-9
Property 18-9

HELPDLG methods 18-9
Destroy dialog 18-9
Initialise help dialog 18-10
Handle keys 18-10
Emphasise 18-10
Make the help dialog visible 18-10

19 Opl and Comms Script Support **19-1**
Precursors 19-1

PROGTRAN 19-1
Class diagram 19-1
Class definition 19-2
Property 19-2

PROGTRAN methods 19-3
Initialise 19-3
Process message 19-3
Start translation of a module 19-3

Deferred PROGTRAN methods 19-3
Get line of source 19-3
Report completion 19-4

PROGEXEC 19-4
Class diagram 19-4
Class definition 19-4
Property 19-5

PROGEXEC methods 19-5
Initialise and queue 19-5
Ensure process terminated 19-6

PROGFIND 19-6
Class diagram 19-6
Class definition 19-6
Property 19-6

PROGFIND methods 19-7
Start the scan 19-7
File system name found 19-7

20 Incremental Matchers **20-1**
Precursors 20-1

MATCHER 20-1
Class definition 20-2
Property 20-2

MATCHER methods 20-2
Handle key input 20-2

Deferred MATCHER methods 20-3
Initialise 20-3
Set current record by match 20-3
Sense current record data 20-3
Set current record 20-3
Sense current record index 20-3
Try clever match 20-3
Handle transition 20-3

VMATCHER 20-4
Class diagram 20-4
Class definition 20-4
Property 20-4

VMATCHER methods 20-5
Destroy 20-5
Initialise 20-5
Set current record by match 20-5
Sense current record data 20-5
Set current record by index 20-5
Sense current record 20-5
Try intelligent match 20-6
Handle transition 20-6
Set range 20-6

WMATCHER 20-7
Class diagram 20-7
Class definition 20-7
Property 20-7

WMATCHER methods 20-8
Destroy 20-8
Initialise 20-8
Set current record 20-8
Try intelligent match 20-9
Handle transition 20-9

WLDSELWN 20-10
Class diagram 20-10
Class definition 20-10
Property 20-11

WLDSELWN methods 20-12
Destroy 20-12
Initialise 20-12
Draw current record 20-12
Set current record 20-12
Handle key input 20-13
Sense data 20-13
Emphasise 20-14
Returns required width 20-14
Set restriction 20-14

21 Link Paste Support 21-1
Precursors 21-1

EWLINKSV 21-1
Class diagram 21-1
Class definition 21-2
Property 21-2

EWLINKSV methods 21-2
Initialise 21-2
Extract some data 21-2

22 Representations of Time and Date 22-1
Precursors 22-1

HTIME 22-1
Class diagram 22-1
Class definition 22-2
Property 22-2

HTIME methods 22-2
Set time 22-2
Set time format 22-2
Get system data 22-3
Set abbreviations 22-4

23 The GATE Class 23-1
Class definition 23-2
Property 23-3

Gate methods 23-5
Check ATS status 23-5
Sense the last keypress 23-5

24 HWIM Utility Functions 24-1

General utilities 24-2
Return TRUE 24-2
Return FALSE 24-2
Destroy an object 24-2
Make a window visible 24-2
Send command to command manager 24-2
Ensure path exists 24-3

Text management 24-3
Allocate cell and load resource 24-3
Load resource into buffer 24-4
Load choice list item into buffer 24-4
Generate error text 24-4
Generate formatted string 24-5
Generate formatted string, variable argument count 24-6
Append ellipsis to text 24-6
Set text mode 24-6
Set font 24-7
Set text style 24-7
Get normal text width for buffer 24-7
Get normal text width for string 24-8
Get bold text width for buffer 24-8

User notification 24-8
Display an information message 24-8
Display an error information message 24-9
Display a busy message 24-9
Make a beep 24-9

Run a dialog 24-10
Launch a dialog 24-10
Run a confirm dialog 24-10
Run a two-line confirm dialog 24-10
Run an error dialog 24-11

Dialog box utilities 24-11
Set an item 24-11
Set a text item 24-12
Set text in edit window 24-12
Set text in prompt window 24-12
Set choice in choice list 24-13
Set On/Off choice list to On 24-13
Set value of numeric editor 24-14
Set latitude/longitude editor 24-14
Set punctuation editor 24-15
Set floating point editor 24-15
Set date editor 24-15
Set range editor 24-16
Set dialog title 24-16
Sense an item 24-17
Sense edit window 24-17
Sense a choice list 24-18
Sense a numeric editor 24-18
Sense a range editor 24-19
Sense a floating point editor 24-19
Sense a latitude/longitude editor 24-19
Sense a punctuation editor 24-20
Sense a date editor 24-20
Dim/undim an item 24-21
Lock/unlock an item 24-21
Change focus 24-21
Set floating point editor from twips value 24-22
Sense twips value from floating point editor 24-22

CHAPTER 1

INTRODUCTION

This manual is a reference document for Psion's HWIM library. It provides a comprehensive guide to the library and documents the classes, methods, properties, inheritance hierarchies and other information essential for understanding and using the library.

It assumes familiarity with the concepts of Object Oriented Programming.

The *Object Oriented Programming Guide* is a useful pre-requisite as it provides the necessary background to Object Oriented Programming as implemented at Psion. It can, of course, be read in conjunction with the *HWIM Reference* manual.

The HWIM library is supplied as the *hwim.dyl* dynamic link library in the ROM of the Series 3 range of machines. It contains a rich set of classes that provide the application management environment and a sophisticated user interface for Series 3 applications.

Access to a subset of the functionality of the HWIM classes is supplied by the non-object oriented HWIF library, described in the *Programming in Hwif* manual. Note that the HWIM and HWIF libraries are mutually exclusive; an application may not use a mixture of the facilities of these two libraries.

The Series 3, Series 3a and Work*about* machines contain slightly differing versions of the HWIM library. This is largely caused by the different screen size and the additional support for the use of grey pixels, but there have also been a number of enhancements. In addition to changes to the internal operation of some methods, the Series 3a version contains additional classes, and some classes that are present on the Series 3 support additional methods on the Series 3a.

The changes, however, are such that code written for the Series 3 will also run on the Series 3a and Work*about*. Conversely, provided some care is taken not to use the enhanced features, code can be written for the Series 3a and Work*about* that will also run on Series 3 machines.

The HWIM software that is supplied with this version of the *SIBO 'C' Software Development Kit* corresponds to the Work*about* version. The content of this manual is also primarily written to describe the Series 3a and Work*about* versions, but differences between the versions for the various machines are noted. In the vast majority of cases, apart from the enhancements, the differences - even where mentioned - are concerned with internal details of implementation and are not significant to application programmers. Note that, except where explicitly mentioned, all changes that are specified for the Series 3a will also apply to the Work*about*.

Many of the HWIM classes contain features that are intended only to be used by system code. Any feature that is not explicitly documented as being usable by application programmers should be considered to be for system use and should not be accessed by application code.

Using HWIM classes

An application (or DYL) that either subclasses or creates an instance of an HWIM class must declare an external reference to the HWIM library (and the OLIB library) in its category file. If, for example, an application's category file has the name *myprog.cat*, the content of this category file must start with the following lines:

```
IMAGE myprog

EXTERNAL olib
EXTERNAL hwim
```

This ensures that, amongst other things, the defined constants representing the external category numbers for the HWIM and OLIB categories (in this case, CAT_MYAPP_HWIM and CAT_MYAPP_OLIB, respectively) are available to application code.

In the source code of the MYPROG application, an instance of an HWIM class - say, of EDWIN - would be created with p_new (or f_new) as follows:

```
p_new(CAT_MYPROG_HWIM,C_EDWIN);
```

.If *myprog.cat* defines a subclass of an HWIM class (say, the class SUBEDWIN) this would exist in the local category. An instance is created using the local category number CAT_MYPROG_MYPROG, as follows:

```
p_new(CAT_MYPROG_MYPROG,C_SUBEDWIN);
```

Similar consideratons apply to instances created by means of f_newsend.

Notation

Names

Except in class diagrams, a class name is always given in upper case, for example WSERV.

The method name in the title line of the description of each method is the defined symbol for the method number, without its leading O_. In the body of the text this name, in lower case letters, is used to refer to the *method function* (or more simply, the *method*) whereas the upper case name refers to the corresponding *message*. Thus, an object's destroy method is executed when the object receives a DESTROY message.

Method function prototypes

The description of each method contains a function prototype that specifies the nature of any return value and the parameters with which the method is called. The parameters exclude the object handle and the method number.

For example, a method for the class WSERV with the title line:

WS_DO_HELP **Launch help system**

and prototyped as:

```
VOID ws_do_help(INT start_id);
```

would be invoked by, for example:

```
p_send3(hand,O_WS_DO_HELP,4);
```

where hand is the handle of an object of the class in question.

This corresponds to a method function declared in C source code as:

```
METHOD VOID wserv_ws_do_help(PR_WSERV *self,INT start_id)
    {
    ...
    }
```

The ✪ symbol

The enter and leave mechanism (which uses p_enter and p_leave) is commonly used to implement structured error recovery. See the *Error Handling and Error Recovery* chapter of the *Object Oriented Programming Guide* and the *Error Handling* chapter of the *PLIB Reference* manual.

Some methods (the vast majority of destroy methods, for example) can never fail and will therefore never call p_leave. The title line of a number of the more significant methods of this type are marked with a leading ✪ symbol.

With the enter and leave mechanism, a call to p_leave should only occur within the protection of a p_enter harness. If p_leave is called outside a p_enter harness, the process will be panicked with panic number 47.

The p_leave mechanism and its use in method functions is discussed briefly in the section on *Structured Error Recovery* later in this chapter.

Class diagrams

To illustrate the inheritance and using relationships between classes, most chapters will contain at least one class diagram.

The notation is a subset of that used by Grady Booch and described in his book *Object-oriented Analysis and Design with applications* (2nd edition) with two minor changes;

- classes which are referenced, but not described, within a chapter (i.e. classes whose full description lies in other chapters of this manual or in a different manual), are underlined,
- the diagrams do not distinguish between 'has' (aggregation) and 'using' (client/supplier) relationships.

Also note that ultimate inheritance from the root class is assumed and is not shown.

Class hierarchy

In understanding the structure of a specific class, remember that methods and property are often inherited from a superclass (or superclasses).

While a class may contain new methods and property, it may also re-define methods inherited from a superclass (or superclasses). Note that methods in a superclass can be what are known as *deferred* methods.

To help illustrate these relationships, each class description in this manual is accompanied by a diagram which shows that class and its superclass(es) in hierarchical order. This diagram is placed at the beginning of the class description.

The diagram consists of a series of adjacent columns. The rightmost column represents the class being described and will be marked by a double line border while the column to its left represents its immediate superclass (if any) marked by a single line border and so on up the hierarchy. Each column is headed by the class name followed by two boxes; the first lists that class's property and the second lists its methods.

Deferred methods are separated from the preceding methods by a blank line and are printed in italics. If a class re-defines an inherited method, the method name in the appropriate superclass is written with a line through it.

For example, the following diagram would be included in a description of class CCCC subclassed from BBBB which itself subclasses AAAA.

AAAA	BBBB	CCCC
property_1 property_2	property_3	property_4 property_5
method_a ~~method_b~~	~~method_b~~ ~~method_c~~ method_d method_e	method_b method_c method_x method_y method_z *method_u*

In this illustration, method_a is supplied by the superclass AAAA, while method_b is replaced in class BBBB and further replaced in class CCCC. Another method, method_c, is introduced in class BBBB but replaced in CCCC, and so on. Note that method_u is a deferred method.

The ROOT class from which all classes are derived is assumed and will not be shown in the diagrams.

Methods and property inherited from a superclass will be described in the appropriate class description.

Structured Error Recovery

As mentioned earlier, the enter and leave mechanism is commonly used to implement structured error recovery.

Use of the p_leave mechanism

In general, you should assume that all methods NOT marked with the ✪ symbol (as discussed in the section on Notation) are capable of calling `p_leave`, even if this is not explicitly mentioned in the method description. In some cases, such as where a method calls, directly or indirectly, a deferred method (which is supplied by a subclasser) it is not possible to specify whether the method may result in `p_leave` being called.

In the event of an error (such as out of system memory) occurring a method may:

- call `p_leave`, passing the (negative) error number,
- return the error number,
- either call `p_leave` or return an error number, depending on the nature of the error.

Some methods call `p_leave(0)`, which has the effect of returning from the `p_enter` harness (with the return value zero) without signalling an error. This is used, for example, to provide a normal exit from a deeply nested function call, without the need for a zero return value to be passed back through the chain of calls. Intermediate functions in the chain may then be declared as `VOID`.

Some method functions that may call `p_leave` are declared as `VOID`. One reason for this may be that the method forms part of a chain, as described in the preceding paragraph. If user code were to send such a message within a `p_enter` harness, the value returned from `p_enter` would be indeterminate if no error arose. The solution is to construct a shell function which sends the message and then returns zero, and call this shell within a `p_enter` harness. The call to `p_enter` will then return either zero (if the method calls `p_leave(0)` or it executes to completion) or a negative error number.

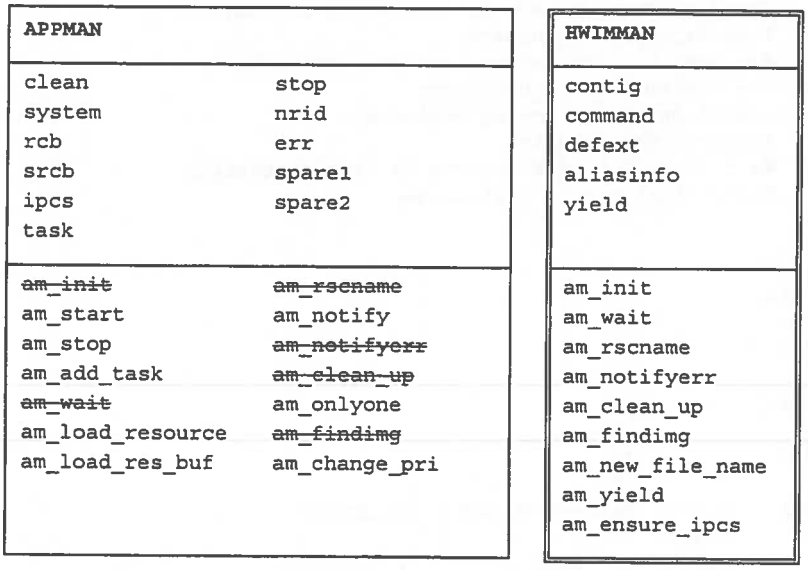
Panic numbers

See the *Error Handling and Error Recovery* chapter of the *Object Oriented Programming Guide* and the *Error Handling* chapter of the *PLIB Reference* manual for a discussion of panics and panic numbers.

HWIM does not have its own unique panic numbers, but panics a client that attempts an illegal operation using PLIB, OLIB and Window Server panic numbers, defined in the respective SDK manuals.

CHAPTER 2

THE HWIMMAN APPLICATION MANAGER



HWIMMAN is a subclass of the OLIB APPMAN class and provides the functionality of the Series 3 HWIM application manager. The APPMAN class is documented in the *OLIB Reference* manual.

Every HWIM application must create and initialise an instance of (a subclass of) HWIMMAN, which must remain in existence until the application terminates. There is no need to send a DESTROY message to an instance of HWIMMAN since its resources will be released, along with all other application resources, on termination of the application.

Subclassers may replace existing methods but, for future compatibility, should avoid adding new methods or property.

As with the APPMAN class, the `ao_run` method of an active object in the application manager's active object queue may terminate prematurely without triggering the error handling mechanism by calling, for example, `p_leave(RUN_ACTIVE_USED)`. In addition, an `ao_run` method has the option of leaving with the call `p_leave(RUN_ACTIVE_CLEANUP_NONOTIFY)`. In this case the cleanup mechanism is triggered but no error notification is presented to the user. This is useful in the case of an error where, for some reason or other, the user is notified of the error before `p_leave` is called.

A built-in application, or any application which supports only one user language, may not need to subclass HWIMMAN. External multi-lingual applications will need at least to replace the `am_rscname` method to load the application resource file.

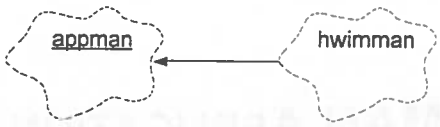
Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the OLIB APPMAN class
- the ACTIVE class (this is also documented in the *OLIB Reference* manual).
- the structure of the Series 3 process command line, as described in the *Series 3 Programming Guide*

- the p_enter and p_leave error handling services

Class diagram



Class definition

Defined in sub-category file *hwimman.cl* (generated header file *hwimman.g*).

```
CLASS   hwimman  appman
Hwim application manager - schedules attached active objects
    {
    REPLACE am_init          Oversee HWIM initialisation
    REPLACE am_wait          Check with window server everything is okay
    REPLACE am_rscname       Name depends on language
    REPLACE am_clean_up      May need to clean wserv temporary resources
    REPLACE am_notifyerr     Use wsAlertW not p_notifyerr
    REPLACE am_findimg       Insist on the SSD being replaced
    ADD am_new_filename      Record a new filename
    ADD am_yield             Wait for all active objects to stop processing
    ADD am_ensure_ipcs       Ensure that ipc is initialised

    CONSTANTS
       {
       H_COMMAND_DEFAULT_FILE          'D'
       H_COMMAND_OPEN_FILE             'O'
       H_COMMAND_CREATE_FILE           'C'
       H_COMMAND_EXIT                  'X'
       H_COMMAND_TRANSLATE_FILE        'T'
       H_COMMAND_RUN_FILE              'R'
       H_COMMAND_LAUNCH_DYL            'L'
       H_COMMAND_BYPASS                'Z'
       FLG_APPMAN_FULLSCREEN   0x1000  Matches PR_WSERV_FULLSCREEN
       FLG_APPMAN_LINKING      0x2000
       FLG_APPMAN_FROM_HWIF    0x4000  Not used by HWIM applications
       FLG_APPMAN_OWNPRIO      0x8000  Matches PR_WSERV_OWNPRIO
       FLG_APPMAN_S3FS         0x0100  Compatibility mode using full screen
       FLG_APPMAN_WSERV_MASK (PR_WSERV_FROM_HWIF|PR_WSERV_OWNPRIO|PR_WSERV_FULLSCREEN)
       RUN_ACTIVE_CLEANUP_NONOTIFY  -1 Same value as E_GEN_FAIL
       }

    TYPES
        {
        typedef struct
            {
            UWORD flags;         flags to supersend
            HANDLE wserv_cat;    cat of wserv
            UWORD wserv_class;   class of wserv
            } IN_HWIMMAN;
        }

    PROPERTY
        {
        UBYTE contig;   TRUE if filename in same alloc block as command line
        UBYTE command;
        TEXT *defext;   The default extension for the application
        TEXT *aliasinfo;
        PR_AIDLE *yield;
        }
    }
```

The Series 3 HWIMMAN class does not support the am_ensure ipcs method. In addition, it does not define the command characters H_COMMAND_LAUNCH_DYL and H_COMMAND_BYPASS, or the FLG_APPMAN_FULLSCREEN flag.

The Work*about* introduces the flag FLG_APPMAN_S3FS.

Property	
hwimman.contig	TRUE if the application's current file name is in the same allocated heap cell as the process command line. This is, if appropriate, set to TRUE by the am_init method, but will be set FALSE by the first use of the am_new_filename method.
hwimman.command	the command byte, if any, read from the process command line by the am_init method (which converts an H_COMMAND_DEFAULT_FILE value to one of H_COMMAND_CREATE_FILE or H_COMMAND_OPEN_FILE, depending on whether the specified file already exists)
hwimman.defext	a pointer to the application's default file extension, if any, as supplied in the process command line. This item is initialised during the am_init method
hwimman.aliasinfo	a pointer to the application's alias information, if any, as supplied in the process command line. This item is initialised during the am_init method
hwimman.yield	either NULL or the handle of an instance of the OLIB AIDLE class. The instance is created automatically on first use of the am_yield method

HWIMMAN methods

AM_INIT — Initialise

```
INT am_init(IN_HWIMMAN *init,UBYTE *wserv);
```

This method is documented for information only. It should not be replaced in any application-specific subclass of HWIMMAN.

Initialise the application, creating an instance of (a subclass of) WSERV together with instances of other classes that may be specified by init->flags. These flags may be a combination of the FLG_APPMAN_XXX flags listed above, together with the following FLG_APPMAN_XXX flags defined for the OLIB APPMAN class:

FLG_APPMAN_CLEAN	Create a cleanup list component
FLG_APPMAN_SYSTEM	Create a system configuration component
FLG_APPMAN_RSCFILE	Create a resource file component
FLG_APPMAN_SRSCFILE	Create a system resource file component
FLG_APPMAN_IPCS	Create an IPCS component
FLG_APPMAN_ONLYONE	Fail if the process already exists

The differences, compared with the allowed flag combinations for the am_init method of the OLIB APPMAN class are as follows:

- An HWIM application must have access to the *system* resource file and so init->flags should include FLG_APPMAN_SRSCFILE (and hence also FLG_APPMAN_CLEAN).
- An HWIM application must be provided with an *application* resource file and so init->flags should include FLG_APPMAN_RSCFILE (and hence also FLG_APPMAN_CLEAN).
- An HWIM application that wishes to create an instance of the SYSTEM class, in order to access the link paste (Bring) system services (described in the *System Services* chapter of the *OLIB Reference* manual) must set init->flags to contain the flag FLG_APPMAN_LINKING rather than the flag FLG_APPMAN_SYSTEM that is required by the APPMAN superclass. If the application wishes to use an instance of the OLIB IPCS class to implement the link paste server, init->flags may contain the flag FLG_APPMAN_IPCS, as for APPMAN. Note that, on all machines except on the Series 3, an instance of IPCS will always be created (but at a later stage in the initialisation) so the presence or absence of the FLG_APPMAN_IPCS flag is less significant on this machine.
- If set, the flag FLG_APPMAN_FULLSCREEN causes applications on the Series 3a and the Work*about* to run in native, rather than Series 3 compatibility, mode.

- The additional flag FLG_APPMAN_S3FS may be set. On the Work*about*, this causes a Series 3 application to run in compatibility mode, but using the full 240x100 screen. To take advantage of this, the Series 3 application must be able to adjust the size of its display to the available screen size.
- The additional flag, FLG_APPMAN_OWNPRIO may be included, to be written to WSERV property (where it is identified as PR_WSERV_OWNPRIO). See the chapter *The WSERV Class* for further details of this flag.

The following operations are performed:

- Writes its own handle to the magic static w_am.
- For the Series 3a, loads the general data that can be accessed via the magic static DatGate.
- Attempts to open a language-specific system resource file, expected to be in the ROM. If this succeeds, the FLG_APPMAN_SRSCFILE bit in init->flags is cleared to prevent any attempt to open the default system resource file by the superclass am_init method.
- Supersends the AM_INIT message to the APPMAN superclass, passing the flag values in init->flags. If the previous step failed to load a language-specific system resource file, this will include an attempt to open the file (as described in the *APPMAN Application Manager Class* chapter of the *OLIB Reference* manual).
- If init->flags contains the flag FLG_APPMAN_LINKING, creates and initialises an instance of the OLIB SYSTEM class for communication with the window server process, *SYS$WSRV*.
- On the Series 3a anf Work*about*, records the current value of the magic static w_ws for later use. This will only be non-zero in the case where the application is started, and hence owned, by another application (see the description of p_getowner in the *Processes and Inter-process Messaging* chapter of the *PLIB Reference* manual). A non-zero value is expected to be the address of a status word in the owning application's data segment.
- Creates an instance of the class init->wserv_class from category init->wserv_cat and writes its handle to the magic static w_ws - it is assumed to be (a subclass of) the WSERV active object class.
- The FLG_APPMAN_OWNPRIO and (for the Series 3a and Work*about*) the FLG_APPMAN_FULLSCREEN flag bits, if present, are extracted from init->flags and written to the w_ws->wserv.flags property field (where they are known as PR_WSERV_OWNPRIO and PR_WSERV_FULLSCREEN respectively).
- On the Work*about*, if the flag FLG_APPMAN_S3FS is present, sets the PR_XWSERV_S3FS flag in DatGate->gate.flags.
- The values of the magic statics DatProcessNamePtr, DatUsedPathNamePtr and DatStatusNamePtr, together with the hwimman.contig, hwimman.defext, hwimman.command and hwimman.aliasinfo property fields, are set up as appropriate from the data in the process command line, pointed to by DatCommandPtr.

  See the example below and *The Series 3 command line* in the *Communicating with the System Screen* chapter of the *Series 3 Programming Guide* for a description of the components of the command line. Note that this processing of the command line is specifically designed to handle a command line of the form that is passed to an application that is started from the System Screen. The command line data passed to an application started in any other way may not be of this form. In such a case (for example, a custom application running on the Work*about*) the application may need to provide its own code to process the command line data.

  On the Series 3a and the Work*about* there is an option to process a reduced command line, signalled by the presence of a command byte with value H_COMMAND_BYPASS. In this case, the command byte is assumed to be immediately followed by the alias information (rather than the ususal public name of the application) and, of the items previously listed, only hwimman.command and hwimman.aliasinfo are initialised. The application is expected to perform its own interpretation or processing of any additional command line data, starting at the byte pointed to by hwimman.aliasinfo.
- If the application is file-based (DatUsedPathNamePtr is not NULL) and the command byte in hwimman->command is H_COMMAND_DEFAULT_FILE, the flag PR_WSERV_CONNECT_AT_BACK is ORed into

w_ws->wserv.flags and the value of hwimman->command is converted to either H_COMMAND_OPEN_FILE or H_COMMAND_CREATE_FILE, depending on whether the file specified by DatUsedPathNamePtr exists or does not exist.

- On the Series 3a and the Work*about*, if the application is not file-based and the command byte is H_COMMAND_BYPASS, the flag PR_WSERV_OWNPRIO is ORed into w_ws->wserv.flags.
- Sends w_ws an AO_INIT message, passing on the wserv parameter (which is a pointer to an IN_WSERV struct).
- On the Series 3a and the Work*about*, sends itself an AM_ENSURE_IPCS message to create an instance of IPCS if one does not already exist.
- Also, on the Series 3a and the Work*about*, creates and initialises an instance of the ATSSV automatic test system server class.
- Enables the future display of a temporary status window by calling wsEnableTemp.
- On the Series 3a and the Work*about*, if the previously stored initial value of w_ws was not zero, it is assumed to be the address of a status word in the data segment of an owning process. A value of zero is written into that status word and the owning process is signalled by means of a call to p_iosignalbypid. This notifies the owner that the owned application has completed its initialisation and is now ready to receive inter-process messages.
- If all the previous steps are completed without error, HWIMMAN sends itself an AM_START message, to start the active object event scheduler. This will not return until the application terminates by sending HWIMMAN the corresponding AM_STOP message. On receipt of this message the application is terminated by means of a call to p_exit. If any of the previous steps result in an error, other than RUN_ACTIVE_CLEANUP_NONOTIFY, the error is reported by means of a call to p_notifyerr before the application is terminated by a call to p_exit.

The method formally returns zero, but the return is never executed since the method always terminates with a call to p_exit. Furthermore, an HWIM application will typically terminate with a direct p_exit(0) call, rather than sending HWIMMAN an AM_STOP message (see, for example, the COMMAN com_exit method in the *Command Manager* chapter).

Although EPOC will not attempt to run the application unless there is sufficient memory for the application's start-up heap (as specified by the *.app* file header - see *Greater control over the image file created* in the *Building an Application* chapter of the *General Programming Manual*) the initialisation could still fail. Possible reasons for failure include the heap space of the window server process or the file server process becoming exhausted, the application being passed the name of a file of the wrong type, or running out of memory while loading a large file.

Command line example

If the heap cell (pointed to by DatCommandPtr) containing the command line for a file-based application has, for example, the following content:

```
ROM::WORD.APP<0><0x29>OProgram<0>.OPL OROPO<0>LOC::M:\WRD\MYPROG.OPL<0>
```

then, at the conclusion of the AM_INIT method:

hwimman.contig is TRUE
hwimman.command is 'O' (H_COMMAND_OPEN_FILE)
DatProcessNamePtr points to the string "Program"
hwimman.defext points to the string ".OPL" (the original following space is overwritten by zero)
hwimman.aliasinfo points to the string "OROPO"
DatUsedPathnamePtr points to the string "LOC::M:\WRD\MYPROG.OPL"
DatStatusNamePtr points to the string "MYPROG.OPL"

For an application that is not file-based, and therefore has nothing following the application file name in its command line, all the items in the above list will be set to NULL.

AM_RSCNAME — Generate resource file name

```
VOID am_rscname(TEXT *pname);
```

Write, to the buffer pointed to by pname, (which must be at least P_FNAMESIZE bytes long) the default full file specification (see the *Files* chapter of the *PLIB Reference* manual) of the application resource file.

The name is generated from the application's start-up full file specification, pointed to by the magic static `DatCommandPtr`. The name also depends on the current language, as determined by the value returned by a call to `p_getlanguage`.

If the language is English (`p_getlanguage` returns a value of 1) the resource file is assumed to be built into the image file, so that the full file specification of the resource file is identical to that of the image file.

For all other languages the resource file is assumed to be located in the same directory and have the same name as the application image file, but with a language-dependent file name extension. The extension is assumed to be *.~nn*, where the characters *nn* represent the language number, as two decimal digits. Thus, a German language resource file would have a *.~03* extension.

This method is designed for use by built-in applications, whose resource files are in the *ROM::* device (which does not support subdirectories) and whose default language is English. A multi-lingual application that is run from an SSD should subclass this method.

The following example, for a `MYHWMAN` subclass of `HWIMMAN`, implements the scheme described in the *Resource Files* chapter of the *Additional System Information* manual. In this scheme the resource file for the default language is built into the image file. Resource files for other languages are located in a subdirectory of the directory containing the application image file. The subdirectory has the same name as the application and the resource file name is derived from the application name, with the final two characters containing the language number.

```
METHOD VOID myhwman_am_rscname(PR_MYHWMAN *self, TEXT *pname)
    {
    TEXT *p;
    P_FPARSE crk;
    P_INFO f;
    TEXT buf[10];

    p_fparse(DatCommandPtr,NULL,pname,&crk);
    p=pname+crk.system+crk.device+crk.path;
    *(p_bcpy(&buf[0],p,crk.name))=0;
    *p++='\\';
    p=p_scpy(p,&buf[0]);
    *p++='\\';
    if (crk.name>6)
        crk.name=6;
    p=p_bcpy(p,&buf[0],crk.name);
    p_atos(p,"%02d.rsc",p_getlanguage());
    if (p_finfo(pname,&f)<0)
        p_scpy(pname,DatCommandPtr);
    }
```

The default resource file, built into the image file, is used if a particular language is not supported by a corresponding resource file.

Of course, in a specific application - where the resource file name is known - less gereral code can be used for this method.

AM_WAIT — Wait for an event

```
VOID am_wait(VOID);
```

Send `w_ws` an `AO_QUEUE` message, to ensure that window server events will be processed (a queued read may have been cancelled) and then call `p_iowait`.

AM_CLEAN_UP — Clean up resources and report an error

```
VOID am_clean_up(INT err,UBYTE *htask);
```

Call `wCleanUp` to free any window server temporary resources, supersend the `AM_CLEAN_UP` message to provide the standard error recovery and reporting, then set `appman.err` to zero.

AM_NEW_FILENAME — Record a new filename

```
VOID am_new_filename(TEXT *newname);
```

Record the new file name pointed to by `newname`. The buffer pointed to by `newname` must remain in existence until the name is changed again.

If the current file name is specified to be in the process command line (`hwimman.contig` is `TRUE`) then the allocated heap cell containing the command line is truncated to remove the name, and `hwimman.contig` is set to `FALSE`.

The magic statics `DatUsedPathNamePtr` and `DatStatusNamePtr` are set up to point to the appropriate positions in the text string.

✪AM_NOTIFYERR — Report an error

```
VOID am_notifyerr(INT err,INT resid);
```

Notify an error for error number `err`, with text as specified by the resource id `resid` providing additional information about the context of the error. This text may be up to 80 bytes in length (50 bytes for the Series 3). If no additional text is required, `resid` may be zero.

This method is guaranteed to succeed in displaying an error message.

Cancels any busy indicator by calling `wCancelBusyMsg` and writes zero to the `WSERV` active object's `wserv.filter` property. On the Series 3a the value of `wserv.filmethod` is also set to zero but, provided the original value of `wserv.filmethod` was negative (ie it is a 'permanent' filter - see the *WSERV Class* chapter) both values are restored after notification of the error. On the Series 3, any application that needs to restore `wserv.filter` to its original state on conclusion of the error report may need to subclass the `am_notify` method.

If `err` has the value `RUN_ACTIVE_CLEANUP_NONOTIFY` the method returns, without displaying any error notification. On the Series 3 the value of `wserv.filter` will have been cleared, but on the Series 3a and the Work*about*, both `wserv.filter` and `wserv.filmethod` are preserved.

Unlike in the superclass `am_notifyerr` method, the `p_notify` service is not used. The method calls the `hErrorDialog` utility function, to attempt to display the error in an error dialog. If this fails due to lack of memory, the error is displayed as an alert, by sending `w_ws` a `WS_ALERT` message. This alert cannot fail, but may not display any context information specified by `resid` if the attempt to load this resource fails.

Using one of these two forms of error display means that the error notification is *application modal*; only that application is suspended and the user may task switch to another application. The superclass method is *system modal*, preventing interaction with *any* application until the user has responded to the error report.

AM_YIELD — Wait for all activity to cease

```
VOID am_yield(VOID);
```

Use an idle active object to suspend the current action until all active objects with priority greater than `PRIORITY_ACTIVE_COMPUTE` have had the opportunity to service their outstanding events.

Automatically creates an instance of the OLIB `AIDLE` class, if it does not already exist, storing its handle in `hwimman.yield`. Sends this object an `AO_QUEUE` message and then sends itself an `AM_START` message, which will not return until the idle object has had an opportunity to run.

✪ AM_FINDIMG — Find application image file

```
INT am_findimg(VOID);
```

Refuse to continue until the image file has been located.

If the image file can not be found, it is assumed that this is because the SSD containing it has been removed. The application is suspended by sending a `WS_ALERT` message to `WSERV`, displaying an alert requesting that the SSD be replaced.

It is worth noting that, in consequence, the `am_load_res_buf` method can never fail in an HWIM application.

AM_ENSURE_IPCS — Guarantee existence of IPCS

```
VOID am_ensure_ipcs(VOID);
```

This method is not available on Series 3 computers.

If it does not already exist, create the application's instance of the IPCS class. The handle of the instance is written to appman.ipcs.

This method is called from the am_init method, following the sending of the AO_INIT message to the applications instance of (a subclass of) the WSERV class.

An application is free to call this method at an earlier stage - for example, from the WSERV ws_dyn_init method.

CHAPTER 3

THE WSERV CLASS

ACTIVE
q
priority
isactive
pcb
stat
destroy
~~ao_init~~
~~ao_cancel~~
ao_abrun
~~ao_queue~~
~~ao_run~~

WSERV	
ws	help_index_id
com	help
dial	lock
bar	subdial
cli	sc[10]
info	locmask
oldinfo	filelist
filter	anim
filmethod	printer
flags	
ao_init	ws_edit_pdev_serup
ao_cancel	ws_edit_print_context
ao_queue	ws_ens_print_context
ao_run	ws_add_filelist
ws_do_dial	ws_remove_filelist
ws_add_dial	ws_anim_tick
ws_remove_dial	ws_switch_files
ws_sense_accel	ws_alert
ws_change_cliwin	ws_unknown
ws_cancel	ws_foreground
ws_wrap_para	ws_background
ws_error_dialog	ws_dyn_init
ws_query_dialog	ws_process_key
ws_do_help	ws_date_changed
ws_do_submenu	ws_launch_dyl
ws_set_menubar	ws_define_fnbar
ws_reset_menubar	ws_hook_today_changed
ws_load_chlist_res	ws_get_alloc_info
ws_append_country	ws_get_print_context
ws_free_dial	ws_self_check
ws_smart_dial	ws_hide_app
ws_dial_env	ws_attach_app
ws_lock	ws_file_info_print
ws_format_dialog	ws_run_memo
ws_evaluate	ws_do_remote_dial
ws_eval_env	ws_user_abandoned
ws_sense_pdev_text	

An instance of the WSERV active object class is the application's event source for events (keypresses, redraws and so on) generated by the window server process. Every HWIM application must create a WSERV object at an early stage in its initialisation and this instance should remain in existence until the application terminates.

The handle of the WSERV object is stored in the w_ws magic static and can be accessed by the associated application.

In addition to its main role as the source of window server events, WSERV supplies a number of general services and utilities, including, for example, methods to:

- start up an application-specific dialog
- run a variety of system-supplied dialogs
- evaluate a numeric expression
- word wrap a paragraph of text

An application is expected to subclass WSERV, at least to supply a `ws_dyn_init` method which should perform all application-specific initialisation.

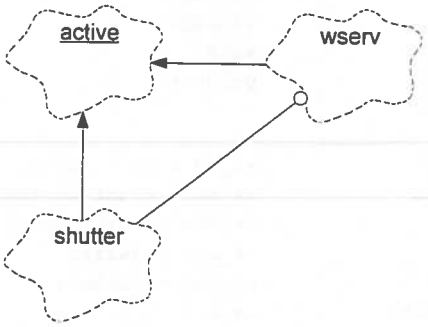
Subclassers may, where appropriate, replace existing methods but, for future compatibility, should avoid adding new methods or property.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the `p_enter` and `p_leave` error handling services
- the OLIB ACTIVE active object class.

Class diagram



Class definition

Defined in sub-category file *hwimman.cl* (generated header file *hwimman.g*).

```
CLASS   wserv   active
The window server active object
    {
    REPLACE ao_init
    REPLACE ao_cancel=p_dummy
    REPLACE ao_queue            Queue a message from the server
    REPLACE ao_run              Process a message from the server
    ADD ws_do_dial              Start a dialog going
    ADD ws_add_dial             Add a dialog to the list
    ADD ws_remove_dial          Remove a dialog from the list
    ADD ws_sense_accel          Lookup an accelerator for a pull down menu
    ADD ws_change_cliwin        Log in a new client window
    ADD ws_cancel               Cancel pending window server read
    ADD ws_wrap_para            Wrapping service
    ADD ws_error_dialog         error dialog
    ADD ws_query_dialog         query dialog
    ADD ws_do_help              Help system
    ADD ws_do_submenu           Run a submenu
    ADD ws_set_menubar          Set an alternative menu bar
    ADD ws_reset_menubar        Reset the menu bar
    ADD ws_load_chlist_res      Get text of a chlist item direct from resource file
    ADD ws_append_country       country selector dialog, for mark up purposes
    ADD ws_free_dial            Invoke the free-form dialling dialog
    ADD ws_smart_dial           smart dial of a number with reference to home
    ADD ws_dial_env             Set or get dial setup environment variable
    ADD ws_lock                 Increment or decrement the lock count
```

```
ADD ws_format_dialog        Launch set format dialog for evaluator or calc
ADD ws_evaluate             Evaluate an expression
ADD ws_eval_env             Set or get evaluator environment variable
ADD ws_sense_pdev_text      Sense text describing current printer device
ADD ws_edit_pdev_setup      Invoke printer device setup dialog
ADD ws_edit_print_context   Invoke print setup dialog
ADD ws_ens_print_context    Check context data has been initialised
ADD ws_add_filelist         Add a filelist to the list
ADD ws_remove_filelist      Remove a filelist from the list
ADD ws_anim_tick            Animator has ticked over
ADD ws_switch_files         Switch files instruction received
ADD ws_alert                Subclassable interface to wsAlert
ADD ws_unknown_wm=p_dummy   Unknown message received from server
ADD ws_foreground=p_dummy   Process WM_FOREGROUND
ADD ws_background=p_dummy   Process WM_BACKGROUND
ADD ws_dyn_init=p_dummy     Applications usually supply this
ADD ws_process_key          Process WM_KEY
ADD ws_date_changed         Process WM_DATE_CHANGED
ADD ws_launch_dyl           Launch DYL in response to wGetCommand('L')
ADD ws_define_fnbar         Utility layer over wsSetList
ADD ws_hook_today_changed   Request or cancel today changed notification
ADD ws_get_alloc_info       p_allcnt facility
ADD ws_get_print_context    Fetch printer context from other process
ADD ws_self_check=p_false   For access by IPC testing or otherwise
ADD ws_hide_app             Hide or show application
ADD ws_attach_app           Appear to attach to other application
ADD ws_file_info_print      Present standard file infoprint
ADD ws_run_memo             Run memo editor
ADD ws_do_remote_dial       Run dialog in attached process
ADD ws_user_abandoned       Exit application with alert

TYPES
    {
    typedef struct
        {
        HANDLE com_cat;     command manager category
        UWORD com_class;    command manager class
        } IN_WSERV;

    typedef struct
        {
        UBYTE menu;             Which menu was pulled down
        UBYTE mnitem;           Which item was selected in the menu
        UWORD menubar_id;       Resource ID of menu bar
        UBYTE first_com;        Method number of first command
        UBYTE count;            Number of accelerators
        UBYTE accel[1];         Accelerator keys
        } WSERV_INFO;     Comes from a resource file

    typedef struct
        {
        UWORD id;           dialog resource ID
        VOID *rbuf;         NULL or address of dialog result buffer
        PR_DLGBOX **pdlg;   NULL or location to receive handle of dialog
        } DL_DATA;

    typedef struct
        {
        TEXT *dupnp;        Stores DatUsedPathNamePtr
        TEXT hidden[6];     For "SYS$X"
        } WS_HIDE_APP_DATA;

    typedef struct
        {
        UWORD margin;       Width in pixels to fill with text
        UWORD fmargin;      Width of margin for first line
        WORD font;          Font used
        UWORD style;        Style used
        WORD nlines;        Max no lines to add to table
        UBYTE *ptable;      Table to write line lengths to
        } WRAP_DATA;
```

```
    typedef struct
        {
        UWORD ncells;
        UWORD nbytes;
        } WS_ALLOC_INFO;

    typedef struct
        {
        VOID *next;
        VOID *object;
        UWORD message;
        } WS_TODAY_HOOK;

    typedef struct
        {
        UWORD len;
        VOID *p;
        } MEMO_DATA_PART;

    typedef struct
        {
        MEMO_DATA_PART body;
        MEMO_DATA_PART styles;
        MEMO_DATA_PART title;
        } MEMO_DATA;

    typedef struct
        {
        VOID *owner;
        UWORD method;
        } MEMO_CALLBACK;
    }

CONSTANTS
    {
    PR_WSERV_CONNECT_AT_BACK        W_CONNECT_AT_BACK    (ie 0x01)
    PR_WSERV_FOREGROUND             0x02
    PR_WSERV_CLIWIN_KEY             0x04
    PR_WSERV_CANCELLING             0x08
    PR_WSERV_BASIC_HELP             0x10
    PR_WSERV_FREEFORM_DIALLING      0x20
    PR_WSERV_INSERT_MODE            0x40
    PR_WSERV_INSERT_PENDING         0x80
    PR_WSERV_OWN_DEFAULT_FONT       0x100
    PR_WSERV_RECEIVED_KEY           0x200
    PR_WSERV_HIDDEN                 0x400
    PR_WSERV_METRIC                 0x800
    PR_WSERV_FULLSCREEN             0x1000
    PR_WSERV_CANCELLING_DIALOG      0x2000
    PR_WSERV_FROM_HWIF              0x4000
    PR_WSERV_OWNPRIO                0x8000
    PR_WSERV_HELP_INDEX             0x01    (Bit re-used)
    WSERV_DTOB_HEX       3          continue from P_DTOB type
    DEGREES_MODE        0x80
    DTOB_MAX_WIDTH      20          -x.xx...x (13 dec places) e-99
    WS_EVAL_ENV_SET         0
    WS_EVAL_ENV_GET         1
    WS_DIAL_ENV_SET         0
    WS_DIAL_ENV_GET         1
    PDEV_TEXT_MAX_LEN       14
    WS_LOCCHG_LOW           0x0100
    WS_LOCCHG_HIGH          0x4000
    WS_EM_STYLES                    0x01
    WS_EM_PRINT_CONTEXT             0x02
    WS_EM_REUSE_BODY_BUFFER         0x04
    WS_EM_REUSE_STYLES_BUFFER       0x08
    WS_EM_CALLBACK                  0x40
    MEMO_NO_CHANGE          0
    MEMO_NULL_LENGTH        1
    MEMO_STANDARD_CHANGE    2
    }
```

```
PROPERTY
    {
    WS_EVENT_X ws;          to take WSERV event
    PR_COMMAN *com;         command manager
    PR_DLGCHAIN *dial;      handle of current dialog, or NULL
    PR_MENUBAR *bar;        menu bar (if present)
    PR_WIN *cli;            client window (if present)
    WSERV_INFO *info;       menu bar ID, command manager cat and class etc
    WSERV_INFO *oldinfo;    store info when in submenu mode
    PR_WIN *filter;         send all keys here if non-NULL
    INT filmethod;          possible filter message (if not WN_KEY)
    UWORD flags;            foreground state, etc
    UWORD help_index_id;    resource ID of application help index
    UWORD help;             count of help screens
    UBYTE lock;             count of how many times locked
    UBYTE subdial;          index of subdialog to launch, less one
    TEXT sc[10];            array of special characters
    UWORD locmask;          mask for locchg
    PR_ROOT *filelist;      topmost filelist
    PR_ROOT *anim;          animator
    PR_ROOT *printer;       print manager
    }
}
```

The following methods, structures and defined constants are not available on the Series 3.

Methods:

```
ws_process_key
ws_date_changed
ws_launch_dyl
ws_define_fnbar
ws_hook_today_changed
ws_get_alloc_info
ws_get_print_context
ws_self_check
ws_hide_app
ws_attach_app
ws_file_info_print
ws_run_memo
ws_do_remote_dial
ws_user_abandoned
```

Structures:

```
WS_HIDE_APP_DATA
WS_ALLOC_INFO
WS_TODAY_HOOK
MEMO_DATA_PART
MEMO_DATA
MEMO_CALLBACK
```

Constants:

```
WS_EM_STYLES
WS_EM_PRINT_CONTEXT
WS_EM_REUSE_BODY_BUFFER
WS_EM_REUSE_STYLES_BUFFER
WS_EM_CALLBACK
MEMO_NO_CHANGE
MEMO_NULL_LENGTH
MEMO_STANDARD_CHANGE
```

The constants `PR_WSERV_HIDDEN` and `PR_WSERV_FULLSCREEN` have replaced the Series 3 constants `PR_WSERV_SHUTTER` and `PR_WSERV_MACRO_DIALOG` respectively. Neither of these two Series 3 constants were used in any Series 3 application code.

Property

WSERV property is accessible via the w_ws magic static. Although this means that the property may be accessed from any point in the application code, the property should, except where write access is specifically allowed, be considered as read-only.

wserv.ws

This, together with the immediately preceding active.stat superclass property, effectively forms a WS_EV struct, in which is stored the data relating to a window server event. The WS_EVENT_X and WS_EV structs are defined in *wlib.h* as:

```
typedef struct
    {
    UWORD handle;   /* Destination handle */
    UWORD time;
    WS_EVENT_UNION u;
    } WS_EVENT_X;

typedef struct
    {
    WORD type;/* E_FILE_PENDING, err or (+ve) event */
    UWORD handle;
    UWORD time;
    WS_EVENT_UNION p;
    } WS_EV;
```

Note that the WS_EV struct and the WS_EVENT struct (also defined in *wlib.h*) are alternative descriptions of the same physical structure. The WS_EVENT_UNION struct is a union of the structures used to store the details of each of the possible window server events (keypress, redraw, etc) and is also defined in *wlib.h*.

wserv.com

The handle of the application's command manager, assumed to be an instance of (a subclass of) COMMAN, set by system initialisation code, in the wserv_ao_init method.

wserv.dial

NULL if the application is not currently displaying a dialog, otherwise the handle of the application's current (foremost) dialog (but see the ws_add_dial method).

wserv.bar

NULL if the application is not displaying a menu bar, otherwise the handle of the application's menu bar, set by system code in the ws_process key and ws_do_submenu methods.

wserv.cli

The handle of the application's client window, set by application-specific initialisation code (in ws_dyn_init) and modified either directly by application code or by the ws_change_cliwin method.

wserv.info

A pointer to a WSERV_INFO struct, containing the the current menu bar and accelerator data. This data is initially loaded from the application resource file during WSERV initialisation of an HWIM application. The elements wserv.info->menu and wserv.info->mnitem respectively indicate which menu was last pulled down and which item was selected in that menu (if both are zero this indicates the first item in the first menu). An application may read (and, if necessary, write to) either or both of these elements.

wserv.oldinfo

A pointer to the main menu bar and accelerator data while an alternate menu or submenu is in use.

wserv.filter

If not NULL, indicates that keyboard input is *filtered*, by being diverted away from its normal destination. The value of wserv.filter may be either -1 to discard all incoming keypresses, or the handle of a window to which all keypresses are diverted. An application may write this property.

wserv.filmethod	If not zero, the method number of the message to be sent to the object specified by wserv.filter on receipt of a keypress. Otherwise a WN_KEY message is sent. On the Series 3a, if the method number is stored in wserv.filmethod as a negative value, the filter is said to be *permanent*. A permanent filter allows keyboard access to the application's Help and is preserved across error notification by the application manager's am_notifyerr method. Note that the concept of a permanent filter does not exist on the Series 3. An application may write this property.
wserv.flags	A collection of state flags, described below.
wserv.help_index_id	Either zero or the resource ID of the application's Help index. An application may write this property.
wserv.help	A count of the number of levels of help currently being displayed, incremented and decremented by system code, in the ws_do_help method and the destroy method of the HELPDLG class.
wserv.lock	Zero if the application is not locked. Otherwise a count of the current number of times the application is locked. The count is incremented and decremented by the ws_lock method.
wserv.subdial	Set and cleared by system code and used to identify a subdialog to be launched by the current dialog. Application code should not modify this item.
wserv.sc	A buffer containing special characters, loaded from the system resource file during WSERV initialisation of an HWIM application.
wserv.locmask	A bitmask used to ensure that successive items in the wserv.filelist list have distinct bits set, in the range WS_LOCCHG_LOW to WS_LOCCHG_HIGH inclusive, for use with calls to p_locchg.
wserv.filelist	Either NULL or the handle of the first item in a list of FILELIST instances.
wserv.anim	Either NULL or the handle of an instance of the OLIB ANIMATOR class, used when wserv.filelist is not NULL to send WSERV a WS_ANIM_TICK message every three seconds.
wserv.printer	The handle of the application's print manager, assumed to be an instance of (a subclass of) the FORM PRINTER class, set by the ws_ens_print_context method.

State flags

The state flags, stored in wserv.flags, may be any combination of the PR_WSERV_XXX flags specified in the class definition (with the exception of PR_WSERV_CONNECT_AT_BACK).

The flags may be divided into several groups. The first group consists of those WSERV flags that may be written into wserv.flags by the HWIMMAN am_init method, before it calls the WSERV ao_init method.

PR_WSERV_OWNPRIO	TRUE if the application is to connect to the window server with the priority specified in the header of its *.app* file. Otherwise the process priority is determined by the window server's client process priority management mechanism. This flag may be set explicitly (as FLG_APPMAN_OWNPRIO) in the flags passed to the application's instance of HWIMMAN from the start-up code in main().
PR_WSERV_FULLSCREEN	On the Series 3a, FALSE if the application is running in compatibility mode. Only applications writtrn expressly for the Series 3a set this flag to TRUE. This flag may be set explicitly (as FLG_APPMAN_FULLSCREEN) in the flags passed to the application's instance of HWIMMAN from the start-up code in main().

PR_WSERV_CONNECT_AT_BACK	TRUE if the application is to connect to the window server as a background process. This flag is exceptional in that it is used during initialisation, but is not stored permanently in wserv.flags - its value is reused for PR_WSERV_HELP_INDEX. The HWIMMAN am_init method includes this flag automatically if the command line includes the command byte H_COMMAND_DEFAULT_FILE. It can not be set explicitly in the flags passed to the application's instance of HWIMMAN from the start-up code in main(). An application that expressly wishes to make a background connection to the window server should replace the WSERV ao_init method to OR PR_WSERV_CONNECT_AT_BACK into wserv.flags and then supersend the AO_INIT message.

The next group consists of those flags used to record semi-permanent state information. These are read and written by system code, but may also be read by application code:

PR_WSERV_FOREGROUND	TRUE if the application is the foreground process.
PR_WSERV_METRIC	TRUE if metric units are selected.
PR_WSERV_RECEIVED_KEY	TRUE if the application has received a key since coming to foreground.
PR_WSERV_HIDDEN	On the Series 3a, this flag indicates whether the application is hidden, being set or cleared by the ws_hide_app method. On the Series 3 it is used internally, under the name PR_WSERV_SHUTTER, in code that controls the locking of an application against Shutdown or Switchfiles messages from the System Screen.

The next group contains flags that are generally of no interest to application code. Most of them indicate states that are more transient than those of the previous group.

PR_WSERV_HELP_INDEX	TRUE if a help index dialog is present.
PR_WSERV_BASIC_HELP	Used during the display of Help to indicate that a 'Help on help' dialog is present.
PR_WSERV_FREEFORM_DIALLING	TRUE if the free-form dialling dialog is present.
PR_WSERV_CANCELLING	Used internally by the mechanism that cancels a read request on the window server process.
PR_WSERV_CANCELLING_DIALOG	Used internally by the dialog cancel mechanism.
PR_WSERV_FROM_HWIF	Must always be FALSE for an HWIM application.

The final group contains flags that are primarily intended for use by an application. Although system code may set or clear them as a service to an application, the application may also write these flags.

PR_WSERV_OWN_DEFAULT_FONT	an application, such as Word, that supplies its own specification of printer fonts, rather than using a single default printer font, should set this flag. It is read by the PRNMODEL dialog and, if set, disables the selection of a default font for printed output
PR_WSERV_CLIWIN_KEY	cleared by the ao_run method when a keypress is sent to a destination other the client window. Used by the Word application to abort the two-character sequence that enters a style or emphasis shortcode

PR_WSERV_INSERT_PENDING	cleared when a keypress event arrives with a keycode of 0x100 or greater, or (but only when using EDWIN) when the keypress is either Psion-Delete or Shift-Psion-Delete. Used by the Word application, together with PR_WSERV_INSERT_MODE, to control the creation of, and insertion into, an emphasis region by entering a two-character shortcode sequence and then typing characters
PR_WSERV_INSERT_MODE	cleared when a keypress event arrives with a keycode of 0x100 or greater, or (but only when using EDWIN) when the keypress is either Psion-Delete or Shift-Psion-Delete. See PR_WSERV_INSERT_PENDING

WSERV methods

WS_DYN_INIT — Application initialisation

```
VOID ws_dyn_init(VOID);
```

The application's WSERV receives a WS_DYN_INIT message at the end of the processing of the ao_init method. The supplied method does nothing.

An application is expected to subclass WSERV to supply a replacement ws_dyn_init method that performs application-specific initialisation.

The application should create and initialise all objects that are required to display the initial view. This will generally involve at least the creation and initialisation of a client window, whose handle should be written to wserv.cli.

An application that has engine components should create and initialise these components either directly from the ws_dyn_init method, or during the initialisation of the client window, depending upon which alternative is more convenient.

A file-based application should open or create a file, depending on the data stored in w_am->hwimman.command and the file name pointed to by DatUsedPathNamePtr. It may also have to take note of the alias information in w_am->hwimman.aliasinfo.

AO_INIT — Initialise

```
VOID ao_init(IN_WSERV *init);
```

This method is documented for information only. It should not, in general, be replaced in any application-specific subclass of WSERV. *One exception to this rule is to make a background connection to the window server, as indicated in the earlier description of the* PR_WSERV_CONNECT_AT_BACK *flag.*

Initialise the WSERV active object.

The following points indicate significant aspects of the initialisation code:

- WSERV adds itself to the application manager's active object queue with priority PRIORITY_ACTIVE_WSERV.
- Special characters (those with H_SC_XXX #defines in *symbols.h*) are loaded into the wserv.sc buffer from the system resource file. Every HWIM application must therefore have access to the system resource file.
- The method loads resource number 1 from the application's resource file into wserv.info. This resource is expected to be a WSERV_INFO resource struct (defined in *hwim.rh*, and duplicating the C WSERV_INFO struct that is declared in the WSERV class definition). Every HWIM application must therefore have a resource file.
- wserv.flags initially contains data written by the HWIMMAN am_init method and thus may contain any combination of PR_WSERV_OWNPRIO, PR_WSERV_CONNECT_AT_BACK. and (on the Series 3a) PR_WSERV_FULLSCREEN. The application allocates memory for a WSERV_SPEC struct and connects to the window server, by calling wConnect, passing the address of this struct and the appropriate combination of W_CONNECT_PRIORITY and W_CONNECT_AT_BACK flags. The call to wConnect writes the address of the WSERV_SPEC struct to the magic static wserv_channel (application code may

subsequently read the contents of the CONNECT_INFO struct accessed by wserv_channel->conn - see the description of wConnect in the *General Window Server Functions* chapter of the *Window Server Reference* manual). The PR_WSERV_CONNECT_AT_BACK flag is removed from wserv.flags, for later reuse as the PR_WSERV_HELP_INDEX flag.

- On the Series 3a, if wserv.flags contains PR_WSERV_FULLSCREEN, the application is switched out of compatibility mode by means of a call to wCompatibilityMode. On the Work*about*, full screen compatibility mode is set if DatGate->gate.flags contains PR_XWSERV_S3FS.
- On the Series 3a the pseudo-constant data that is accessed via the DatGate magic static is set up (including DatGate->gate.x and, additionally for the Work*about*, DatGate->gate.dx - see the chapter: *The GATE Class*).
- An instance of a command manager, whose class and category are specified by the IN_WSERV struct pointed to by init, is created and sent a COM_INIT message. All HWIM applications must therefore have a command manager, which will normally be a subclass of COMMAN.
- If a call to p_getctd indicates that the machine is currently set to use metric units, PR_WSERV_METRIC is ORed into wserv.flags.
- The final action of the ao_init method is to send a WS_DYN_INIT message. The application *must* subclass WSERV to supply this method.

AO_QUEUE — Queue a message from the server

```
INT ao_queue(VOID);
```

HWIM applications are not expected to replace or to make explicit use of this method.

If there is already an outstanding read (note that the HWIMMAN am_wait method always sends WSERV an AO_QUEUE message) flush the client-side buffer by calling wCheckPoint. If this were not done window server calls (for example, wInvalidateRect) could, in some circumstances, remain buffered for an indefinite period of time. Regularly flushing such calls ensures that any consequential WM_REDRAW event is received promptly and the screen's appearance is kept up to date.

Otherwise, queue a read to the window server process by calling wGetEvent and then setting active.isactive to TRUE. The call sets active.stat to E_FILE_PENDING. On completion of the read, the result will be written to active.stat and the immediately following wserv.ws (these two items of property effectively form a WS_EV struct, described earlier in this chapter).

For historical reasons the method returns zero.

AO_CANCEL — Cancel

```
VOID ao_cancel(VOID);
```

HWIM applications are not expected to replace this method.

Does nothing. The WSERV cancel functionality is performed by the ws_cancel method, described below.

The reason for this is that an active object's ao_cancel method is called from the object's destroy method, but the WSERV cancel action is not appropriate when WSERV is being destroyed.

To some extent this is no longer necessary since the current tendency is that an application never destroys its instance of WSERV, relying on operating system code to recover an application's resources when the application terminates.

WS_CANCEL — Cancel any pending window server read

```
VOID ws_cancel(VOID);
```

HWIM applications are not expected to replace this method.

Cancel any pending read queued on the window server process by calling wCancelGetEvent.

Since the destruction of a window involves operations on both the client side and the server side of application/window server interaction, it is possible that the window server could send an event, such as a redraw event, destined for a window that is in the process of being destroyed. To prevent this possibility,

the ws_cancel method is called at an early stage of any window destruction sequence called from within the run method of an active object (apart from WSERV itself). *This is done automatically when required and it is not expected that an application will ever have to make explicit use of this method.*

Note that there is no need to explicitly requeue a request following a cancel since WSERV is sent an AO_QUEUE message by the HWIMMAN am_wait method.

AO_RUN — Process a message from the server

```
INT ao_run(VOID);
```

HWIM applications are not expected to replace this method.

Handle an event that indicates the completion of a window server read - the application has received an inter-process message from the window server process. On entry to the method the event type is in active.stat and any further data relating to the event is in wserv.ws. In particular, if relevant, the handle of the destination window is in wserv.ws.handle.

The ao_run method always returns RUN_ACTIVE_USED.

If the event is not one of those listed below, WSERV sends itself a WS_UNKNOWN_WM message. Otherwise the various event types are handled as follows:

WM_REDRAW

Sends the destination window a WN_REDRAW message, passing the address of wserv.ws.u.rect, specifying the rectangle to be redrawn.

WM_FOREGROUND

If wserv.flags contains PR_WSERV_HIDDEN, indicating that the application has attached to another process (this mechanism is not available on the Series 3) that process - provided it has not terminated - is brought to the foreground by means of a call to wClientPosition. Otherwise, processing is as follows.

Clears PR_WSERV_RECEIVED_KEY and set PR_WSERV_FOREGROUND in wserv.flags. If wserv.anim is not NULL, it is sent an AO_QUEUE message with a time interval of zero, to restart any animated message. WSERV then sends itself a WS_FOREGROUND,TRUE message.

WM_BACKGROUND

Clears PR_WSERV_FOREGROUND in wserv.flags. If wserv.anim is not NULL, it is sent an AO_CANCEL message to suspend any animated message. WSERV then sends itself a WS_BACKGROUND,FALSE message.

WM_COMMAND

Processes an Exit or a Switchfiles message from the System Screen. Does nothing if the magic static DatLocked is non-zero.

If DatLocked is zero, calls the window server function wGetCommand. If the content of the first byte of the resulting command buffer is H_COMMAND_EXIT, the HWIM utility function hWservComSend is used to send the command manager a COM_EXIT message. Otherwise the command string is assumed to contain a file name and WSERV sends itself a WS_SWITCH_FILES message, passing the address of the (temporary) command buffer.

WM_CANCELLED

There is no action on receipt of this event.

WM_KEY

If DatGate->gate.getkeys is not NULL, it is the handle of an instance of ATSSV. This object is notified of the receipt of a keypress by being sent an SV_KEY message (see the *Series 3a Automatic Test System* chapter of the *Object Oriented Programming Guide*, and the description of ATSSV in the *Automatic Test System Classes* chapter of the *XADD Reference* manual).

WSERV sends itself a WS_PROCESS_KEY message, passing the values of wserv.ws.u.key.keycode and wserv.ws.u.key.modifiers.

Note that, on the Series 3, the ws_process_key method does not exist. On the Series 3, processing equivalent to that described for the ws_process_key method is performed by code that is called directly from the ao_run method.

WM_DATE_CHANGED

WSERV sends itself a WS_DATE_CHANGED message.

This event will not occur on the Series 3.

WS_PROCESS_KEY — Process WM_KEY

```
VOID ws_process_key(INT keycode, INT modifiers);
```

This method is not available on the Series 3.

On the Series 3, equivalent processing is performed by code that is called directly from the ao_run method. There are minor differences between the code executed in the two cases and, where significant, the differences are noted in the following description. Note that the Series 3 W_KEY_MODE keycode is broadly treated in the same way as described for W_KEY_DIAMOND.

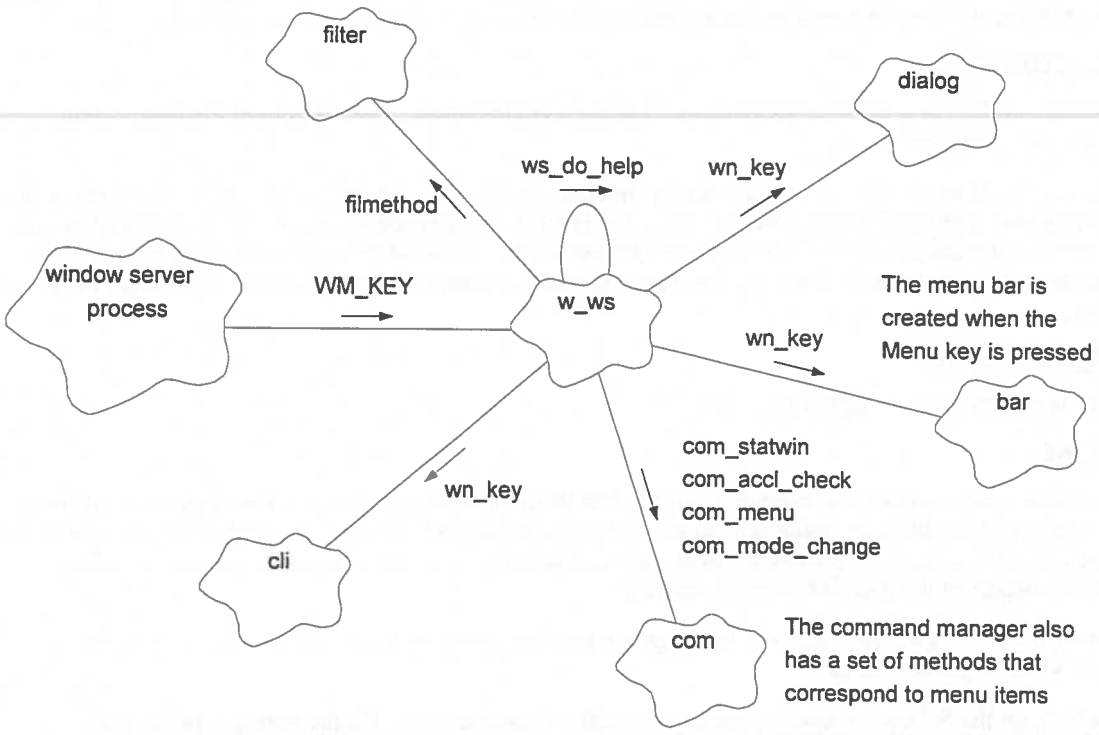
Unless otherwise stated, the processing of all types of keypress terminates with the clearing of the PR_WSERV_CLIWIN_KEY flag in wserv.flags.

The method first clears the W_CAPS_MODIFIER flag in modifiers since the caps lock status is of no significance to HWIM key processing. It sets the PR_WSERV_RECEIVED_KEY in wserv.flags to record that a keypress has been received since the last time the application became the foreground process.

There is some further manipulation of the keycode and modifiers values to aid the distinguishing and recognition of command accelerators. If keycode is W_KEY_DIAMOND and modifiers contains W_PSION_MODIFIER, the value W_SPECIAL_KEY is ORed into keycode. If modifiers contains W_CTRL_MODIFIER, the W_SPECIAL_KEY flag is removed from keycode, regardless of the value it contains; no keypress will be recognised as an accelerator if the Control key is held down.

If the keypress is not consumed by a keyboard filter, as described below, and if keycode is greater than 0xff (cursor movement keys and others - see the *Events* chapter of the *Window Server Reference* manual for a full list of the keys concerned) the PR_WSERV_INSERT_MODE and PR_WSERV_INSERT_PENDING bits are cleared in wserv.flags, as a service to the Word application (see the earlier description of these flags).

The keypress may be dispatched to one of a number of destination objects, as illustrated in the following diagram, where the clockwise order of the destinations broadly indicates their priorities. The destination depends on both the type of the keypress and the presence or absence of the various destination objects, as described below.



Keyboard filters

Keyboard input may be *filtered*, that is, diverted from its normal destination, by setting a non-zero value for `wserv.filter`. The filter behaviour may be further modified by the value of `wserv.filmethod`.

If `wserv.filter` is equal to -1, regardless of the value of `wserv.filmethod`, the keypress is discarded and processing of the keypress terminates.

Any other non-zero value of `wserv.filter` is assumed to be an object handle and a message is generally sent to that object. The only exception is that a permanent filter (`wserv.filmethod` is negative) does not filter the Help key, nor does it filter any key if Help is being displayed. (Note that this exception does not apply to the Series 3, which does not support the concept of a permanent filter.)

If the absolute value of `wserv.filmethod` is not zero it is assumed to be the method function number of the message to send to `wserv.filter`, otherwise a `WN_KEY` message is sent. In all cases the message contains the parameters `keycode` and `modifiers`. The message may return a value of `WN_KEY_CHANGED` to indicate that the keypress has been 'consumed' and that no further processing is required. For any other return value, the keypress is further processed as described below, as if `wserv.filter` were `NULL`.

If `wserv.filter` is `NULL`, or if the message sent to `wserv.filter` returns any value except `WN_KEY_CHANGED`, the action depends on the current state of the application and on the values of `keycode` and `modifiers`, as described in the following sections.

Help

If keycode is `W_KEY_HELP`, and Help is not currently being displayed, one of the following is done, depending on the value of `modifiers`:

`Help`	The resource ID of the help to be displayed is found by sending a `WN_SENSE_HELP` message. The message is sent to either the current dialog or the client window, depending on whether a dialog is present (`wserv.dial` is not `NULL`) or absent. The resulting ID is sent as a parameter to a `WS_DO_HELP` message, to display the appropriate Help, and processing of the keypress then terminates.
`Control-Help`	Provided `wserv.flags` does not contain `PR_WSERV_HELP_INDEX`, sends a `WS_DO_HELP` message to display the system Help index and then terminates processing of the keypress. Otherwise the behaviour is as for the `Help` key.
`Control-PSION-Help` (`Control-Dial`)	Sends a `WS_FREE_DIAL` message to run the free format dialling dialog and then terminates processing of the keypress.
`other`	Continues with further processing.

Dialogs

If a dialog is present (`wserv.dial` is not `NULL`) a `WN_KEY` message is sent to the dialog, passing keycode and modifiers. The `WN_KEY` message is not sent under two circumstances:

- if `modifiers` contains the `W_SPECIAL_KEY` flag, processing of the keypress simply terminates.
- if the dialog's `win.flags` contains `DLGCHAIN_WITH_MENU`, processing continues as described in the following section, to allow access to the dialog's command menu. Any keypress that is not consumed by the command and command menu processing will be offered back to the dialog via a `WN_KEY` message. Note that this option is not available on the Series 3.

If the return value from the `WN_KEY` message is non-zero, the dialog is sent a `DESTROY` message. If the return value is `WN_KEY_CANCELLED`, the value that will be returned by the `WS_DO_DIAL` method that launched the dialog is set to zero. If, on return from the `WN_KEY` message, `wserv.subdial` is non-zero, the dialog is sent a `DL_LAUNCH_SUB` message, passing the index of the dialog item that is launching the subdialog (one less than the value of `wserv.subdial`) and `wserv.subdial` is cleared. Processing of the keypress then terminates.

Commands and command menus

The value of keycode is then tested for any value that initiates a command by one of the following three means:

- A value of `W_KEY_DIAMOND` (`W_KEY_MODE` on the Series 3) is interpreted as a `COM_MODE_CHANGE` command. On machines other than the Series 3, the `COM_MODE_CHANGE` message is sent direct to the

command manager, passing the value of modifiers masked with W_SHIFT_MODIFIER. On the Series 3 the message is sent as described below, with a preceding COM_ACCL_CHECK message (which is expected always to return TRUE in this case).

- If the W_SPECIAL_KEY bit is set in keycode (indicating the Psion key was held down) this bit is masked out and keycode is tested against the list of accelerators held in wserv.info->accel. A match determines the method function number of the command to be executed, otherwise processing of the keypress terminates. The match is forced to be independent of case on the Series 3 but is otherwise case-dependent.
- If a menu bar is displayed, it is sent a WN_KEY message, passing keycode and modifiers. If this returns a zero value, processing of the keypress terminates. Otherwise, the return value specifies the method function number of the command to be executed.

Before executing the command (with the exception of COM_MODE_CHANGE, as noted above) a COM_ACCL_CHECK message is sent to the command manager, passing the proposed method function number. Processing of the keypress terminates at this pont if the com_accl_check method returns FALSE.

Otherwise, if a menu bar is displayed, it is sent a DESTROY message and the underlying window is sent a WN_EMPHASISE,TRUE message. This window is normally the client window but, except on the Series 3, it could be a dialog that allows the use of a command menu.

The command is executed by sending the command manager the appropriate message, by means of the HWIM hWservComSend utility function (again, with the exception of COM_MODE_CHANGE). Processing of the keypress then terminates.

If keycode has a value of W_KEY_MENU, one of the following is done, depending on the value of modifiers:

Menu	Create an instance of MENUBAR, storing its handle in wserv.bar, load the menu bar resource with resource ID wserv.info->menubar_id and send the menu bar a WN_INIT message, followed by a WN_VISIBLE,WV_INITVIS message. Finally, send the client window a WN_EMPHASISE,FALSE message and terminate processing of the keypress.
Shift-Menu	As for the Menu key.
Control-Menu	Send the command manager a COM_STATWIN message, by means of the HWIM hWservComSend utility function, and terminate processing of the keypress.
other	Terminate processing of the keypress.

Client window

Any unconsumed combination of values of keycode and modifiers is sent to the client window in a WN_KEY message. On return from this message, processing of the keypress terminates, *without clearing the* PR_WSERV_CLIWIN *bit in* wserv.flags. Thus, this bit is only set if the keypress has been offered for processing to the client window. Any return value from the client window is ignored.

If a dialog is in existence, any keypress not consumed by the command and command menu processing is offered to the dialog, as described earlier, rather than to the client window.

WS_CHANGE_CLIWIN — Log a new client window

```
PR_WIN *ws_change_cliwin(PR_WIN *wh);
```

Provided that the window with handle wh is not already the current client window, record this window as the new client window.

The previous client window, which must still exist, is de-emphasised (it is sent a WN_EMPHASISE,FALSE message). The new client window is emphasised (sent a WN_EMPHASISE,TRUE message) and is made the foreground window. Its handle is stored in wserv.cli.

Returns the handle of the previous client window as a courtesy to the caller.

The method does nothing but return the client window handle if the new window handle is the same as the previous one.

✪ WS_SENSE_ACCEL — Look up a menu accelerator

```
INT ws_sense_accel(UINT comid);
```

HWIM applications are not expected to replace this method.

Return the lower case accelerator character corresponding to the command manager method number `comid`. The return value is meaningless if the method number is outside the range for which accelerators are defined.

This method is used by the display code for pull-down menus.

WS_DO_DIAL — Run a dialog

```
INT ws_do_dial(HANDLE cat, INT class, DL_DATA *pdata);
```

Load, initialise and run the dialog specified by category handle `cat` and class number `class`. All HWIM dialogs must be started, directly or indirectly, via this method.

The method creates an instance of the class specified by `cat` and `class`, assumed to be (a subclass of) DLGBOX. It then loads the resource specified by `pdata->id` and uses it to create and initialise the dialog's components. The DL_DATA struct is defined in the WSERV class definition section of *hwimman.g* as:

```
typedef struct
    {
    UWORD id;           dialog resource ID
    VOID *rbuf;         NULL or pointer to dialog result buffer
    PR_DLGBOX **pdlg;   NULL or location to receive dialog handle
    } DL_DATA;
```

On the Work*about*, if the flags in the dialog's resource contain DLGBOX_SMALL_FONT, this flag is cleared in the loaded resource and the dialog's `dlgbox.font` property is set to DLGBOX_ROMAN8_FONT.

The dialog is then sent a DL_INIT message, followed by DL_DYN_INIT and DL_SET_SIZE messages. The initialisation of the dialog is completed by a call to the HWIM utility function `hInitVis` and this completion is noted by ORing PR_WIN_INITIALISED into the dialog's `win.flags`. If `pdata->pdlg` is not NULL, the dialog's handle is written to `*pdata->pdlg`. The dialog is then added to the `wserv.dial` list of dialogs by means of a WS_ADD_DIAL message.

If the dialog's `dlgbox.flags` does not contain DLGBOX_NO_WAIT, the application manager is sent an AM_START message. This will not return until the dialog has terminated.

Users of this method should note the following significant points:

- The dialog receives, in order, DL_INIT, DL_DYN_INIT, and DL_SET_SIZE messages before being made visible.
- The creation and initialisation is protected from out of memory failure, by the use of OLIB CLEANUP mechanisms, until the dialog has been made visible. The application does not need to provide any explicit protection against failure unless application initialisation code allocates additional resources.
- The dialog handle is not lodged in `*pd->pdlg` until the dialog has been made visible.

The return value is zero if the dialog is cancelled by the cancel mechanism provided by system code (that is, without the intervention of application-specific code - see the *Dialog Boxes* chapter for further details). This value is only of significance for dialogs for which the DLGBOX_NO_WAIT flag is *not* set.

The method calls `p_leave` (which will trigger the automatic cleanup mechanism) on failure.

✪ WS_ADD_DIAL — Add a dialog to the dialog list

```
VOID ws_add_dial(PR_DLGCHAIN *hand);
```

Add the object with handle `hand` (normally a dialog but, more accurately, an instance of any subclass of DLGCHAIN) to the front of the list whose first element is pointed to by `wserv.dial`.

If a menu bar is visible send it a DESTROY message, otherwise send a WN_EMPHASISE, FALSE message to the currently emphasised window, which will be either the first item in the `wserv.dial` list or, if this list is

empty, the client window. Then add `hand` to the front of the list (`wserv.dial` now contains the handle `hand`) and send this object a `WN_EMPHASISE,TRUE` message.

Note that system-supplied subclasses of `DLGCHAIN` include help screens, `DLGBOX` and all system dialog classes.

✪ WS_REMOVE_DIAL — Remove a dialog from the dialog list

```
VOID ws_remove_dial(PR_DLGCHAIN *hand);
```

Remove the dialog with handle `hand` (not necessarily the first in the list) from the list of dialogs whose first element is pointed to by `wserv.dial`. If, after its removal, there are no dialogs in the list, `wserv.dial` will now be `NULL`. Then send a `WN_EMPHASISE,TRUE` message to either the first dialog in the list or, if there are no such dialogs, the client window.

WS_WRAP_PARA — Paragraph word wrap

```
INT ws_wrap_para(TEXT *buf, INT len, WRAP_DATA *pd);
```

Word wrap the first `len` characters in the buffer pointed to by `buf`, writing the number of characters in each line to successive bytes of the table pointed to by `pd->ptable` (assumed to be `pd->nlines` bytes long).

The `WRAP_DATA` struct is defined in the `WSERV` class definition as:

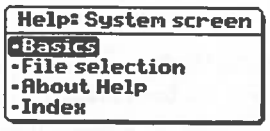
```
typedef struct
    {
    UWORD margin;   width (pixels) to fill with text
    UWORD fmargin;  width (pixels) of first line
    WORD font;      font used
    UWORD style;    style used
    WORD nlines;    max no lines to add to table
    UBYTE *ptable;  table of bytes to receive line lengths
    } WRAP_DATA;
```

Returns either the number of lines into which the text has been wrapped, or zero if there is not enough space in the line length table.

WS_DO_HELP — Run help system

```
VOID ws_do_help(INT start_id);
```

Create, initialise and make visible a help dialog displaying the help information contained in the resource with ID `start_id`.



Create an instance of the `HELPDLG` class and intialise by sending it a `WN_INIT` message with an argument of `start_id`. Make the dialog visible and add to the dialog list by sending a `WS_ADD_DIAL` message to self passing as argument the handle of the `HELPDLG` instance. Increment `wserv.help` and `OR` `PR_WIN_INITIALISED` into the `win.flags` property of the `HELPDLG` instance: this indicates that the `HELPDLG` destroy method should decrement `wserv.help`.

WS_LOAD_CHLIST_RES — Get choice list resource text

```
TEXT *ws_load_chlist_res(INT rid, INT nsel, TEXT *buf);
```

Copy, into the buffer pointed to by `buf`, the zero terminated string that forms choice list item number `nsel` (0 selects the first item) in the choice list resource with ID `rid`. This method is somewhat analogous to the more general application manager `am_load_res_buf` method.

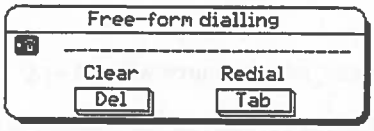
It is the user's responsibility to ensure that `rid` refers to a choice list resource and that the buffer is sufficiently long to contain the specified string.

The method returns a pointer to the trailing zero that follows the loaded resource.

WS_FREE_DIAL — Run the free-form dialling dialog

```
VOID ws_free_dial(VOID);
```

Create, initialise and make visible the free-form dialling dialog:



provided it is not already present and the application is not currently displaying a Help dialog.

WS_SWITCH_FILES — Process a Switchfiles message

```
VOID ws_switch_files(UBYTE *buf);
```

Create an instance of the SHUTTER active object class and send it an AO_INIT message, passing buf, which is assumed to point to a buffer containing a command byte followed by a string specifying the new file name.

This method is called as a result of the receipt of a Switchfiles message from the System Screen. If necessary, the SHUTTER active object delays the processing of the message (this processing includes sending the command manager a COM_SWITCH_FILES message) until the application is in a suitable state to proceed.

See the description of the SHUTTER class, later in this chapter, for a fuller explanation of Switchfiles processing.

WS_LOCK — Alter the lock count

```
INT ws_lock(INT lock);
```

Add or remove a level of locking, depending on whether lock is TRUE or FALSE.

The application is considered to be locked if the magic static DatLocked is non-zero. A locked application does not receive Exit or Switchfiles messages from the System Screen, that is, the WSERV ao_run method ignores events of type WM_COMMAND.

Note that the method stores the locked state in both wserv.lock and DatLocked.

It is the programmer's responsibility to ensure that adding and removing levels of locking are balanced within an application.

WS_SET_MENUBAR — Set alternative menu bar

```
WSERV_INFO *ws_set_menubar(INT rid);
```

Use hLoadResource to allocate a cell and load into it the resource with ID rid (assumed to be a WSERV_INFO resource that defines the accelerators and the menu bar resource ID) writing the address of the cell to wserv.info.

Returns the previous value of wserv.info, pointing to the allocated memory holding the original menu bar resource data. It is the programmer's responsibility to store this value for future restoration of the original menu bar data (normally by use of ws_reset_menubar). If, exceptionally, this resource is not to be restored, it may be released with a call to p_free.

Note that this method does not make the new menu bar visible, but the new menu will appear when the user next presses the Menu key. There is no straightforward automatic means of forcing the new menu bar to be displayed.

WS_RESET_MENUBAR — Reset the menu bar

```
VOID ws_reset_menubar(WSERV_INFO *info);
```

Free the heap cell pointed to by wserv.info and set wserv.info equal to info.

This is typically used as follows:

```
p_send3(w_ws,WS_RESET_MENUBAR,w_ws->wserv.oldinfo);
```

to restore the main menu data after, for example, using the `ws_set_menubar` method.

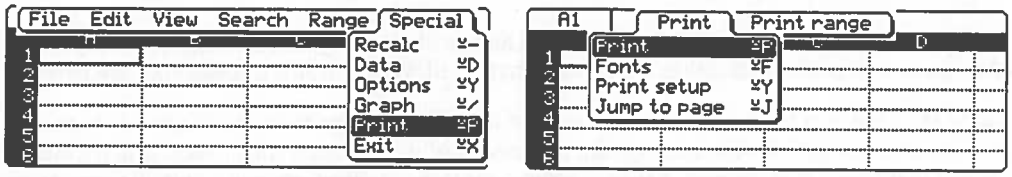
WS_DO_SUBMENU — Run a submenu

```
VOID ws_do_submenu(INT rid);
```

Run the submenu specified by the `WSERV_INFO` resource with ID `rid`.

Sends a `WS_SET_MENUBAR` message, passing `rid`, storing the returned pointer (to the loaded main menu resource data) in `wserv.oldinfo`. Then creates, initialises and makes visible an instance of `MENUBAR` (writing its handle to `wserv.bar`) to display the specified submenu. Sends a `WN_EMPHASISE,FALSE` message to the client window.

An example of a submenu is the Series 3 Spreadsheet application's *Print* submenu, displayed by selecting the *Print* command from the *Special* menu of the main menu bar:



Note that, as illustrated above, commands in a submenu may have accelerators that duplicate those appearing in the main menu. Submenus therefore provide a means of exceeding an application's normal limit of 30 commands (see *The Command Manager*).

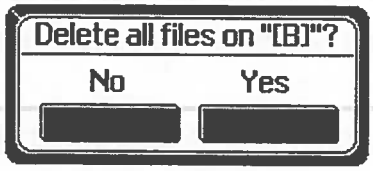
This method would normally be called from within the command manager method associated with the main menu option that gives rise to the submenu.

The original menu bar is automatically restored when the submenu menu bar ceases to be visible.

WS_QUERY_DIALOG — Run a query dialog

```
INT ws_query_dialog(INT secondrid, INT rid, INT *pargs);
```

Run the system query dialog, with up to two lines of text.



The first line of text is generated, using `hAtob`, from the format string loaded from the resource with ID `rid` and the list of arguments pointed to by `pargs`. The generated text may not exceed 50 characters, including the terminating zero. The value of pargs may be `NULL` if there are no arguments, and `rid` may be zero, in which case there will be no first line text. The second line of text is loaded from the resource with ID `secondrid`, which is assumed to be a simple string resource. The value of `secondrid` may be zero, in which case there will be no second line text.

On the Series 3 this method can call `p_leave` if there is insufficient memory to create and initialise the dialog. On the Series 3a the method will not fail, since it will call `wsAlertW` to present an equivalent alert if there is insufficient memory to display the dialog.

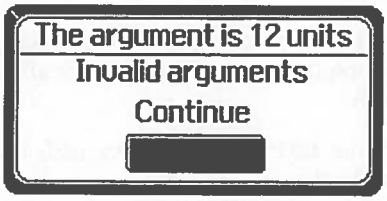
The method returns `TRUE` if the user confirms, otherwise it returns `FALSE`.

See also the `hConfirm` and `h2LineConfirm` utility functions.

WS_ERROR_DIALOG — Run an error dialog

```
INT ws_error_dialog(INT err, INT rid, INT *pargs);
```

Run the system error dialog, with up to two lines of text.



The first line of text is generated, using `p_errs`, from the error number `err`. The second line of text is generated, using `hAtob`, from the format string loaded from the resource with ID `rid` and the list of arguments pointed to by `pargs`. The generated text may not exceed 50 characters, including the terminating zero. The value of pargs may be `NULL` if there are no arguments, and `rid` may be zero, in which case there will be no second line text.

Returns zero, to confirm that the method did not call `p_leave`. This method is suitable for calling under the protection of `p_enter`.

See also the `hErrorDialog` utility function.

WS_EVALUATE — Evaluate an expression

```
INT ws_evaluate(TEXT *pResult, TEXT *pExpr, VOID *oplmod);
```

Evaluate the expression contained in the zero terminated string pointed to by `pExpr`, writing the result to the buffer pointed to by `pResult`.

The result is evaluated according to the format preferences derived from the evaluator environment variable, *M$V*, accessed via the `ws_eval_env method`, described below.

If the initial value of the byte `*pResult` is zero, the evaluation is performed in so-called 'calculator' mode. After evaluation in this mode, the result buffer contains the numerical value, as a `DOUBLE`, in its first eight bytes. This is followed by the text representation of the value, as a zero terminated string, using the *calc* preferences from *M$V*.

Any initial non-zero value in `*pResult` causes the evaluation to be performed in 'evaluator' mode. After evaluation in this mode, the result buffer contains only the text representation of the value, as a zero terminated string, using the *eval* preferences from *M$V*.

The expression to be evaluated may be any expression that is acceptable to the OPL programming language. The value of `oplmod` may be either `NULL` or a pointer to the name of an OPL module containing additional functions (for example, a set of hyperbolic trigonometrical functions) to be used in evaluating the expression.

Returns -1 if the evaluation is successful. Otherwise reports an error, using `hInfoPrintErr`, and then returns the byte offset, within the buffer at `pExpr`, to the point at which the error was detected.

WS_EVAL_ENV — Set or get evaluator environment variable

```
VOID ws_eval_env(EXTENDED_MEM_VALUES *pev, INT getit);
```

Write, from `*pev`, or read, to `*pev`, the *M$V* environment variable, which stores evaluator format preferences.

The `EXTENDED_MEM_VALUES` struct is defined in *h_eval.h* as:

```
typedef struct
    {
    UBYTE evalFormat;  /* Dtob format code for all except calc */
    UBYTE evalDPlaces; /* Decimal places for all except calc */
    UBYTE calcFormat;  /* Dtob format code */
    UBYTE calcDPlaces; /* Decimal places, if relevant */
    DOUBLE values[MAX_MEMORIES];
    } MEM_VALUES;
```

```
typedef struct
    {
    UBYTE evalDegrees;
    UBYTE calcDegrees;
    MEM_VALUES memVal;
    } EXTENDED_MEM_VALUES;
```

Note that the Series 3 supports two global sets of evaluator preferences, one specifically for the Calculator application (set by the Calculator's *Format* command) and one for all other evaluations (set by the System Screen's *"Evaluate" format* command).

If `getit` is `TRUE`, read the contents of the *M$V* environment variable into `*pev`. If the environment variable does not exist it is created with default values as follows:

```
pev->evalDegrees=DEGREES_MODE;
pev->calcDegrees=DEGREES_MODE;
pev->memVal.evalFormat=P_DTOB_FIXED;
pev->memVal.evalDPlaces=EVAL_DEFAULT_PLACES;
pev->memVal.calcFormat=P_DTOB_GENERAL;
pev->memVal.calcDPlaces=CALC_DEFAULT_PLACES;
p_bfil(&pev->memVal.values[0],MAX_MEMORIES*sizeof(DOUBLE),0);
```

If `getit` is `FALSE`, write `*pev` to the *M$V* environment variable. Note that this process modifies the contents of `*pev`.

WS_DIAL_ENV — Set or get dial environment variable

```
VOID ws_dial_env(DIAL_ENVAR *pev, INT getit);
```

Write, from `*pev`, or read, to `*pev`, the *D$X* environment variable, which stores telephone dialling preferences.

The `DIAL_ENVAR` struct is defined in the `SDIALDLG` smart dialling dialog class definition as:

```
typedef struct
    {
    UWORD toneLengthTicks;
    UWORD delayLengthTicks;
    UWORD pauseLengthTicks;
    UBYTE dialOutCode[6];       to access an external line
    } DIAL_ENVAR;
```

If `getit` is `TRUE`, read the contents of the *D$X* environment variable into `*pev`. If the environment variable does not exist it is created with default values as follows:

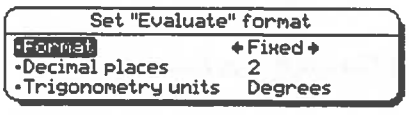
```
pev->toneLengthTicks=8;
pev->delayLengthTicks=8;
pev->pauseLengthTicks=48;
p_scpy(&pev->dialOutCode[0],"9,"); /* from the SYS_DIAL_OUT system resource */
```

If `getit` is `FALSE`, write `*pev` to the *D$X* environment variable.

WS_FORMAT_DIALOG — Run the set format dialog

```
VOID ws_format_dialog(INT flags);
```

Run the system dialog to set the evaluation format preferences.



If flags is `TRUE`, run the dialog to set the general evaluate preferences, as shown in the above diagram, otherwise run it to set the preferences for the calculator.

On exiting the dialog, except by pressing Esc, the preferences are written to the *M$V* environment variable. See the description of the `EVALDLG` system dialog class for further details.

WS_ALERT — Interface to wsAlert

```
VOID ws_alert(TEXT *t1, TEXT *t2);
```

Ensure the magic static DatLocked is TRUE and then call:

```
wsAlertW(WS_ALERT_CLIENT,t1,t2,0,0,0);
```

On return from this call, DatLocked is restored to its original value.

This method is called from the HWIMMAN am_notifyerr method as the final fail-safe stage of the system's error reporting mechanism.

WS_APPEND_COUNTRY — Run country selector dialog

```
INT ws_append_country(TEXT *outStr);
```

Run the dialog used to add the country name to a telephone number (on pressing Control-Shift-PSION-Help) when editing a Data application entry:



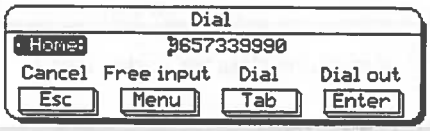
Writes, as a zero terminated string, the selected name, enclosed in square brackets and preceded by a single space, to the buffer pointed to by outStr and returns TRUE if the dialog is terminated by pressing Enter. Otherwise just returns FALSE.

See the description of the CNTRYDLG system dialog class for further details.

WS_SMART_DIAL — Smart dial of a number

```
VOID ws_smart_dial(SMART_DIAL_DATA *data);
```

Run the smart dialling dialog:



to dial any of up to four (on the Series 3) or six (on the Series 3a and Work*about*) telephone numbers passed in the SMART_DIAL_DATA struct pointed to by data. The struct is defined in the SDIALDLG class definition, effectively as:

```
#define  SMART_DIAL_MAX_PROMPT   11 /* max prompt length */

typedef struct
    {
    TEXT pmt[SMART_DIAL_MAX_PROMPT+1];
    TEXT str[WR_MAX_IN_STRING+2];
    } SMART_DIAL_ITEM;

typedef struct
    {
    WORD count;          how many separate numbers
    SMART_DIAL_ITEM it[4];
    } SMART_DIAL_DATA;
```

where WR_MAX_IN_STRING is defined in *wr_io.h*.

See the description of the SDIALDLG system dialog class for further details.

WS_ENS_PRINT_CONTEXT Ensure print context data exists

```
VOID ws_ens_print_context(VOID);
```

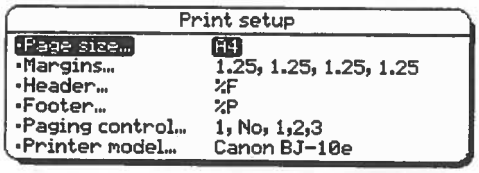
If it does not already exist, create and initialise an instance of the FORM PRINTER class, writing its handle to wserv.printer.

The method does nothing if wserv.printer is not NULL.

WS_EDIT_PRINT_CONTEXT Run print setup dialog

```
VOID ws_edit_print_context(VOID);
```

Send a WS_ENS_PRINT_CONTEXT message and then run the print setup dialog as, for example, is used by the *Print setup* command in the Series 3 Word application's *Special* menu:

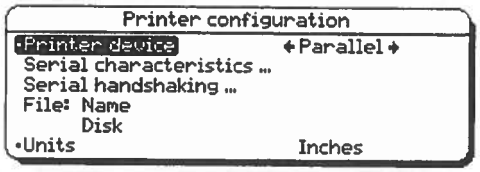


This dialog is used to review and/or modify the print context data stored in the application's instance of the PRINTER class, whose handle is in wserv.printer. See the *Print Classes* chapter for further details of the *Print setup* dialog.

WS_EDIT_PDEV_SETUP Run printer configuration dialog

```
VOID ws_edit_pdev_setup(VOID);
```

Run the dialog to set up the system-wide printer device configuration, as for the *Printer setup* command in the Series 3's System Screen's *Special* menu:



See the description of the PDEVDLG system dialog class for further details.

WS_SENSE_PDEV_TEXT Sense text for current printer device

```
INT ws_sense_pdev_text(TEXT *buf);
```

On the Series 3, this is a VOID *function.*

Write as a zero terminated string, to the buffer pointed to by buf, the text that describes the current printer device. This text will be one of "Parallel", "Serial" or, if printing to file, the name and extension of the print file.

Except on the Series 3, returns the ID of the current printer device.

WS_ADD_FILELIST Add a filelist to the list

```
VOID ws_add_filelist(PR_FILELIST *fl);
```

Add the instance of FILELIST with handle fl to the front of the list whose first item is pointed to by wserv.filelist.

The instance's property, fl->filelist.locmask is set to a distinct single bit value in the range WS_LOCCHG_LOW to WS_LOCCHG_HIGH inclusive

If wserv.anim is NULL, the method creates and initialises an instance of the ANIMATOR class, writing its handle to wserv.anim. The AO_INIT message sets up the ANIMATOR instance to send WSERV a WS_ANIM_TICK message every 3 seconds, with an initial delay of 6 seconds.

✪ WS_REMOVE_FILELIST Remove a filelist from the list

```
VOID ws_remove_filelist(PR_FILELIST *fl);
```

Remove the instance of FILELIST with handle fl from the list whose first item is pointed to by wserv.filelist.

If, after the removal, the list is empty and wserv.anim is not NULL, a DESTROY message is sent to the object with this handle and wserv.anim is set to NULL.

The method does nothing if fl does not appear in the list.

WS_ANIM_TICK Animator has ticked over

```
VOID ws_anim_tick(VOID);
```

If it exists, send the object whose handle is stored in wserv.filelist an FL_LOCCHG message.

WS_UNKNOWN_WM Unrecognised WSERV event

```
VOID ws_unknown_wm(VOID);
```

This message is received when an event of a type not known to the ao_run method is received from the window server process. At the time of writing, possible unrecognised event types are WM_ON and WM_TASK_UPDATE (this latter event type will be received only by the System Screen).

The supplied method does nothing.

An application should subclass this method if it needs to process such messages.

WS_FOREGROUND Foreground message

```
VOID ws_foreground(UINT flag);
```

The application receives this message, with flag set to TRUE, when it becomes the foreground process. The supplied method does nothing.

The flag parameter is passed so that a subclasser may map the ws_foreground and ws_background methods to a single method function, the two cases being distinguished by the value of flag.

WS_BACKGROUND Background message

```
VOID ws_background(UINT flag);
```

The application receives this message, with flag set to FALSE, when it becomes a background process. The supplied method does nothing.

The flag parameter is passed so that a subclasser may map the ws_foreground and ws_background methods to a single method function, the two cases being distinguished by the value of flag.

WS_HOOK_TODAY_CHANGED Request date change notification

```
VOID ws_hook_today_changed(VOID *handle, INT message);
```

This method is not available on the Series 3.

Register or de-register the interest of the object specified by handle in being notified of the arrival of a WM_DATE_CHANGED inter-process message.

If message is non-zero, the interest is registered by storing handle and message at the front of a list of such data items. On receipt of a WM_DATE_CHANGED inter-process message, message number message will be sent to the object specified by handle.

If message is zero, the entry for the object indicated by handle is removed from the list.

It is normally a programming error to register the same object more than once. An attempt to remove an object that has not been registered is a programming error with unpredictable results.

WS_DATE_CHANGED — Process WM_DATE_CHANGED

```
VOID ws_date_changed(VOID);
```

This method is not available on the Series 3.

Send a message to each object that has registered, via the `ws_hook_today_changed` method, interest in the occurrence of date changes.

Each object that has such an interest is sent a message of the type specified at the time it registered its interest.

The `WS_DATE_CHANGED` message is sent from the `ws_process_key` method, in response to a `WM_DATE_CHANGED` inter-process message from the Window Server process.

WS_LAUNCH_DYL — Launch a DYL

```
VOID ws_launch_dyl(TEXT *pname);
```

This method is not available on the Series 3.

Load and link the DYL whose full file specification is pointed to by `pname`, create an instance of its first class and send this instance a message with message number 1 (by convention, this is an initialisation message). This method is indended for use by an application that is started with a command line that contains the command byte `H_COMMAND_LAUNCH_DYL` (`'L'` - see the *HWIMMAN Application Manager* chapter).

If the loading of the DYL fails because the DYL is has already been loaded by this application, the `ws_launch_dyl` method simply returns, without reporting any error. Any other loading error causes `p_leave` to be called.

If the DYL is successfully loaded and linked, an instance of its class number zero is created. This instance is sent, under the protection of `p_enter`, message number 1, passing the DYL's category handle as the method's sole parameter. The corresponding method is expected to return zero to indicate its successful conclusion.

If the method terminates without error, the DYL is considered to be successfully launched and `ws_launch_dyl` then returns. If the method returns a non-zero value, or calls `p_leave`, the DYL's instance of class number zero is sent a `DESTROY` message and the DYL is unloaded. Any such error is not reported.

If successfully launched, it is the DYL's responsibility to ensure that it unloads itself at some later time, when its task is complete. A convenient means of ensuring that the DYL is unloaded correctly is to supply the DYL's class number zero with a component object that is an instance of the XADD `UNLOAD` class (using `PROPERTY n` in the class definition so that the component will receive a `DESTROY` message when the owning class is destroyed). This component should be created and initialised as the last step of the initialisation of its owning class. See the description of the `UNLOAD` class in the *Additional Active Object Classes* chapter of the *XADD Reference* manual.

WS_DEFINE_FNBAR — Set up a status window 'diamond' list

```
INT ws_define_fnbar(INT resid, INT pos, UBYTE *pselect);
```

This method is not available on the Series 3.

Set up, from a resource file, the text for the list of modes to be displayed in the application's status window. This method provides a utility layer over the Window Server's `wsSetList` function

The text is read from the `MENU` resource with ID `resid`. The following example shows the resource used by the Series 3a Agenda:

```
RESOURCE MENU diamond_list
    {
    items =
        {
        MENU_ITEM { mn_item="Day"; },
        MENU_ITEM { mn_item="Week"; },
        MENU_ITEM { mn_item="Year"; },
        MENU_ITEM { mn_item="To-do"; },
        MENU_ITEM { mn_item="Anniv"; },
        MENU_ITEM { mn_item="List"; }
        };
    }
```

Note that there is a fairly serious width restriction on the text of each item - if you use more than five characters, you should test the application carefully.

As for `wsSetList`, the value of `pos` should be either `W_STATUS_WIN_NO_DIAMOND` or the index, counting from zero, of the item against which the diamond symbol is to be displayed.

If `pselect` is `NULL`, all the text items from the resource will be shown in the list. Otherwise, `pselect` should point to a `UBYTE` array where each byte is either `TRUE` or `FALSE` to include or exclude the corresponding text item.

The following example code uses the resource listed earlier. It would set up the Agenda diamond list to show no diamond symbol and contain the two text items "Week" and "Anniv".

```
LOCAL_C VOID SetDiamondItems(VOID)
    {
    UBYTE select[6];

    p_bfil(&select[0],6,FALSE);
    select[1]=TRUE;
    select[4]=TRUE;
    p_send5(w_ws,O_WS_DEFINE_FNBAR,DIAMOND_LIST,W_STATUS_WIN_NO_DIAMOND,&select[0]);
    }
```

The method calls `p_leave` if any error occurs. It returns zero if successful and is,therefore, suitable for being called under the protection of `p_enter`.

WS_GET_ALLOC_INFO — Display allocator statistics

```
VOID ws_(VOID);
```

This method is not available on the Series 3.

Walk the application's heap to determine the number of allocated cells and the total number of bytes of allocated memory.

The method presents an information message showing both these values.

An application may send this message to itself or the message may be sent from an external process, as part of an automated test suite for the application.

WS_GET_PRINT_CONTEXT — Copy print context from a process

```
INT ws_get_print_context(UWORD pid);
```

This method is not available on the Series 3.

Copy the print context from the process with ID `pid`.

The method first sends a `WS_ENS_PRINT_CONTEXT` message to ensure that the application has default print context data. It then performs a series of inter-process copies to overwrite the application's print context data with the corresponding data from the specified process.

It is assumed that the process with ID `pid` exists and has a valid print context.

The method calls `p_leave` if any error occurs during the copying of the data. If the method completes successfully it returns zero to indicate that `p_leave` has not been called. The method is suitable for being called under the protection of `p_enter`.

WS_SELF_CHECK — Perform internal data check

```
INT ws_self_check(VOID);
```

This method is not available on the Series 3.

Perform an internal self-consistency check on the application's data.

The supplied method simply returns FALSE which, by convention, implies success of the check. This method is intended to be replaced to provide application-specific consistency checks.

An application may send this message to itself or the message may be sent from an external process, as part of an automated test suite for the application. The behaviour following the failure of a self-check will be application-specific.

WS_HIDE_APP — Hide application from System Screen

```
VOID ws_hide_app(UWORD pid, WS_HIDE_APP_DATA *p);
```

This method is not available on the Series 3.

Hide the application from, or reveal it to, the System Screen.

This method is intended to be used as part of the mechanism to attach one application to another (see the *Series 3a Attached Applications* chapter of the *Object Oriented Programming Guide*). It should be called by the controlling application, after it has successfully launched the attached process.

The application is hidden if `pid` is non-zero. In this case, the value of `pid` is assumed to be the process ID of the attached process. The application's current `DatUsedPathNamePtr` is preserved in `p->dupnp` and `DatUsedPathNamePtr` is set to point to the string "SYS$X" (a name that will not be displayed in the System Screen's file lists). Provided the attached process has not yet terminated, the application is made system modal and sent to the back of the task list by means of a call to `wSystemModal`. The value `PR_WSERV_HIDDEN` is ORed into `wserv.flags`.

If `pid` is zero, the application is restored to visibility. The application's original value of `DatUsedPathNamePtr` is recovered from `p->dupnp` and, by means of a call to `wCancelSystemModal`, the application's modal state is cancelled and it is brought to the front of the task list. Finally, the value `PR_WSERV_HIDDEN` is cleared from `wserv.flags`.

Note that the `WS_HIDE_APP_DATA` struct must remain in existence until the application is restored to visibility.

WS_ATTACH_APP — Set attached state

```
VOID ws_attach_app(UWORD pid);
```

This method is not available on the Series 3.

Set the application's appearance as being attached to the application with process ID `pid` (the process that controls the attachment (see the *Series 3a Attached Applications* chapter of the *Object Oriented Programming Guide*). It should be called by the attached application itself, during its initialisation.

Sets `DatLocked` to the value of `DatLocked` in the controlling process and then sets `DatProcessNamePtr` to point to a copy of the text pointed to by `DatProcessNamePtr` in the controlling process.

Copies over the text pointed to by `DatUsedPathNamePtr` in the controlling process and sends the application manager an `AM_NEW_FILENAME` message, passing a pointer to this text.

Any error will result in `p_leave` being called.

WS_FILE_INFO_PRINT Display a file-related message

```
VOID ws_file_info_print(TEXT *fname, INT rid);
```

This method is not available on the Series 3.

Present a file-related information message.

The text of the message is constructed from the file specification pointed to by `fname` and the text string resource with resource ID `rid`. The string resource is expected to be a format string containing a single `%s` that will be replaced by the file name.

The passed file specification is parsed to locate the file name and extension. If the extension matches the application's default extension, pointed to by `w_am->hwimman.defext`, the extension is removed from the file name. The file name is string capitalised and the message is displayed by means of a call to the `hInfoPrint` HWIM utility function, passing `rid` and a pointer to the file name (with or without a trailing extension).

The method is used extensively by applications built into the Series 3a to report the completion of file-based operations, in conjunction with the resource strings:

```
SRT_FILE_CREATED
SRT_FILE_OPENED
SRT_FILE_SAVED
SRT_FILE_COMPRESSED
SRT_FILE_MERGED
```

Thus, for example:

```
GLREF_D VOID *w_ws;

TEXT *p;

p="LOC::M:\AGN\AGENDA.AGN";
p_send4(w_ws,O_WS_FILE_INFO_PRINT,p,-SRT_FILE_SAVED);
```

will, in the Agenda application, display the message:

```
"Agenda" saved
```

WS_RUN_MEMO Run Agenda memo editor

```
INT ws_run_memo(INT flags, MEMO_DATA *pd, MEMO_CALLBACK *cb);
```

This method is not available on the Series 3.

This method is supplied with the specific intention of being used by the Agenda application to edit a Memo attached to an Agenda entry. It is not intended to be used by any other application.

WS_DO_REMOTE_DIAL Run dialog in another process

```
VOID ws_do_remote_dial(PR_ACTIVE **pa, INT mainrid, INT butrid);
```

This method is not available on the Series 3.

Run a dialog (an instance of ATSDIAL) in either an attached process or the foreground process.

Determines the process ID of an attached process or, if no such process exists, the current foreground process, and checks that this process is suitable for receiving the instruction to run the dialog.

The method then loads the main dialog resource, specified by the resource ID `mainrid`. If `butrid` is not zero, the 'button' resource that it specifies is also loaded from the resource file. This resource may be *either* an ACLIST_ARRAY, defining the buttons for the dialog's single instance of an action list dialog item, *or* a MENU resource, defining the contents of the dialog's single instance of a choice list dialog item.

If the dialog is not being run in an attached process, the value of `pa` must be NULL. Otherwise it should be a pointer to memory into which a handle can be written. In this case the `ws_do_remote_dial` method runs the dialog under the protection of an idle object (an instance of the OLIB AIDLE class) whose handle is stored in `*pa` as a service to the caller. This idle object will receive an AO_RUN message on successful termination of the dialog.

An inter-process message starts the remote dialog.

Providing there has been no error, the method does not terminate until the remote dialog is complete, when it returns the key code that terminated the dialog. On error, the method calls `p_leave`.

Note that, although ATSDIAL allows an option to present an editable text control in the dialog, the `ws_do_remote_dial` method does not support this mechanism.

WS_USER_ABANDONED — Exit application with alert

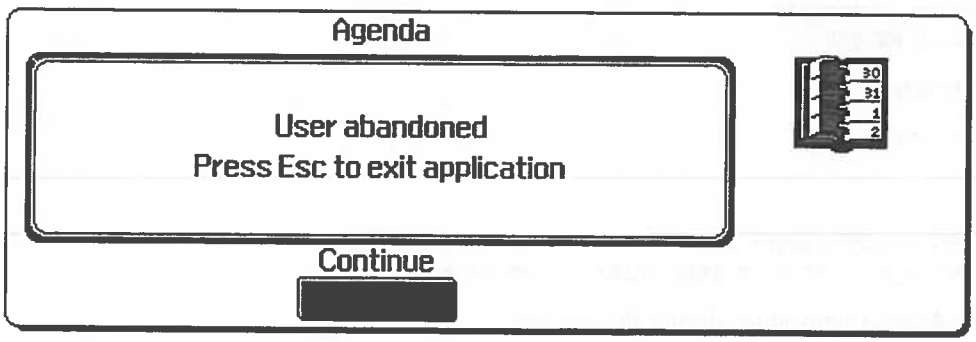
```
VOID ws_user_abandoned(VOID);
```

This method is not available on the Series 3.

Provide a standard means of terminating an application, with user notification.

The method first terminates any attached process. It then presents an alert displaying the two messages 'User abandoned' and 'Press Esc to exit application'.

This method is intended to be used in the event that the user abandons a crucial operation, leaving the application in a state where it is unable to continue. The following example is taken from the Agenda application. It is displayed, for example, when a user removes the SSD containing the current Agenda file, attempts to modify an Agenda entry and refuses to replace the SSD whrn prompted to do so.



SHUTTER

active
q priority isactive pcb stat
destroy ~~ao_init~~ ao_cancel ~~ao_abrun~~ ao_queue ~~ao_run~~

shutter
count buf
ao_init ao_run ao_abrun

An instance of the SHUTTER class is used by WSERV to implement a switch to a new file initiated by a Switchfiles message from the System Screen. It is not expected that an HWIM application will ever explicitly create an instance of SHUTTER, nor is it ever expected to send explicit messages of any kind to any instance. The SHUTTER class is documented as a way of explaining the mechanism of the processing of a Switchfiles message, and under what circumstances that processing may succeed or fail.

Processing a Switchfiles message

A Switchfiles message may be received by an application at any time that it has not set itself in the locked state (see the WSERV `ws_lock` method). Even if an application is not locked, it may not be in a state where it can immediately respond to the message. It may be displaying a menu bar, or it may be interacting with the

user via a dialog. (which is likely to manipulate data that will change when a new file is loaded). The action of the SHUTTER object is to attempt, by removing any displayed menu and trying to cancel any dialogs, to return the application to a state in which it can respond to the Switchfiles message.

A Switchfiles message is received by WSERV as an event of type WM_COMMAND to the ao_run method. This, in turn, results in WSERV being sent a WS_SWITCH_FILES message, whose main task is to create and initialise an instance of SHUTTER.

SHUTTER's first action is to remove any visible menu bar. Then, in successive calls of its ao_run method, it attempts to close down all the items in the WSERV wserv.dial list. These items, most commonly dialogs, are subclasses of DLGCHAIN and are expected to conform with the guidelines for this class. In some cases an item may not fully conform (for example, where a dialog responds to the Escape key by presenting an "Are you sure?" query dialog) and SHUTTER may fail to close such an item. On such a failure SHUTTER destroys itself, the net result being that the Switchfiles message is not processed.

Otherwise, when all the wserv.dial items have been closed down, SHUTTER's final action, before destroying itself, is to process the Switchfiles message by sending the command manager a COM_SWITCH_FILES message.

Class definition

Defined in sub-category file *hactive.cl* (generated header file *hactive.g*).

```
CLASS shutter active
    {
    REPLACE ao_init
    REPLACE ao_run
    REPLACE ao_abrun
    CONSTANTS
        {
        SHUTTER_BUFFER_LEN      128
        SHUTTER_COUNT_MAX       32
        }
    PROPERTY
        {
        WORD count;
        UBYTE buf[128];
        }
    }
```

Property

shutter.count	a count of the number of 'hits' of the SHUTTER ao_run method, used to terminate SHUTTER if it has not been able to initiate the file switch in 32 cycles
shutter.buf	holds the Switchfiles command byte followed by a zero terminated string containing the name of the new file

SHUTTER methods

AO_INIT — Initialise

```
VOID ao_init(UBYTE *buf);
```

Copy SHUTTER_BUFFER_LEN bytes from the buffer pointed to by buf into shutter.buf. This data consists of a Switchfiles command byte followed by a zero terminated file name. If a menu bar is displayed (w_ws->wserv.bar is not NULL) destroy the menu bar and send the client window (with handle w_ws->wserv.cli) a WN_EMPHASISE,TRUE message.

Then add itself to the application manager's active object queue with priority PRIORITY_ACTIVE_WSERV-1 and send itself an AO_QUEUE message.

AO_RUN — Do, or prepare for, the file switch

```
INT ao_run(VOID);
```

If `shutter.count` is equal to SHUTTER_COUNT_MAX, send itself a DESTROY message and return, otherwise increment `shutter.count`.

If a 'dialog' is visible (`w_ws->wserv.dial` is not NULL) send itself an AO_QUEUE message to ensure the later receipt of a further AO_RUN message. Then send `w_ws->wserv.dial` a WN_KEY message with a keycode of W_KEY_ESCAPE. If this message returns a non-zero value, indicating that the dialog has not destroyed itself, send the dialog a DESTROY message. This process will be repeated in successive `ao_run` methods until all items in the `w_ws->wserv.dial` list have been cancelled and destroyed unless the number of attempts exceeds SHUTTER_COUNT_MAX.

Otherwise, if `w_ws->wserv.dial` is NULL, the command manager (`w_ws->wserv.com`) is sent, under the protection of `p_enter`, a COM_FILE_CHANGE message, passing the command byte and a pointer to the file name, both of which are stored in `shutter.buf`. On an error-free return from this message, indicating that the file switch has been successfully completed, the method sends itself a DESTROY message and returns. Any error results in `p_leave` being called.

Returns RUN_ACTIVE_USED.

AO_ABRUN — Handle error

```
VOID ao_abrun(VOID);
```

Supersend the AO_ABRUN message to perform standard error reporting and then send itself a DESTROY message.

CHAPTER 4

THE COMMAND MANAGER

The command manager is a component of the application's instance of (a subclass of) WSERV. Every HWIM application must create and initialise a command manager, which must remain in existence for the lifetime of the application. The command manager may be accessed via its handle which is stored in `w_ws->wserv.com`. The category and class of the application's command manager are passed in an IN_WSERV struct to the application manager's `am_init` method from the application's `main`, as described in the *Introduction* chapter.

There is no need to send a DESTROY message to a command manager since its resources will be released, along with all other application resources, on termination of the application.

Any HWIM application that supports commands that are initiated by a selection from a menu bar and pull-down menu (or a corresponding accelerator) must subclass COMMAN to ADD a method for each such command (with the exception of the Exit command command that is assumed to be handled by the (possibly replaced) `com_exit` method. These methods must be ADDed in a single uninterrupted group, following immediately after the superclass `com_exit` method, in the command manager's class definition.

It is not, however, always necessary to supply a separate method function for each method. All invocations of command manager methods that may be selected from a menu bar (that is, `com_exit` and the following methods added by a subclass) are via the `hWservComSend` utility function, which sends messages to the command manager, passing the command method number as an additional parameter. The command manager's class definition may, if it is convenient, assign two or more methods to share a common method function. This function can distinguish between the different cases by the method number in this parameter.

Every command must have an associated accelerator. These accelerators must be listed in the first item of the application resource file, in the same order as the corresponding command manager methods in the class definition. However, this is not necessarily the order in which they appear in the application's command menus; note that the *Exit* command, corresponding to the `com_exit` method (which is necessarily the first method in the sequence) conventionally appears as the last item in the last menu of all Series 3 applications. See also the *Commands and Command Menus* and *HWIM Resource Files* chapters of the *Object Oriented Programming Guide*.

All Series 3 keyboards, irrespective of language differences, support 30 distinguishable accelerator keypresses (the 26 non-accented alphabetic keys are common to all machines, but the remaining four depend on the keyboard layout, which varies from language to language). As a result, a Series 3 HWIM application is nominally restricted to a maximum of 30 commands, including *Exit*. This limit may, however, be exceeded if the application supports alternative menus and/or sub-menus.

Series 3a machines distinguish between upper and lower case accelerators and therefore may use up to 56 separate commands in a single menu bar.

Each alternative menu or submenu requires a further contiguous block of ADDed command manager methods and a further resource file item to provide an accelerator for each command.

Precursors

Familiarity with the following topics would be helpful:

- the `p_enter` and `p_leave` error handling services
- the WSERV class, especially the `ao_init`, `ao_run` and `ws_switch_files` methods, all of which send messages to the command manager

Class diagram



COMMAN

COMMAN
com_init com_statwin com_accl_check com_menu com_mode_change com_file_change com_exit

The COMMAN class provides the basic skeleton for a command manager, supplying minimal functionality for the methods that an HWIM command manager must support.

Although it is possible to build an application that uses an instance of COMMAN as its command manager (for example, an application that consists of nothing but a chained sequence of dialogs) any non-trivial application will subclass COMMAN, replacing one or more of the supplied methods and adding further application-specific methods. All file-based applications must replace the `com_file_change` method and, if they can modify the contents of their current file, the `com_exit` method.

Class definition

Defined in sub-category file *comman.cl* (generated header file *comman.g*).

```
CLASS   comman  root
Superclass of all command managers
    {
    ADD com_init=p_dummy            Users own initialisation
    ADD com_statwin=p_dummy         Toggle permanent status window
    ADD com_accl_check=p_true       Called whenever an accelerator is matched
    ADD com_menu=p_dummy            Called whenever a menu is pulled down
    ADD com_mode_change=p_dummy     W_KEY_MODE received
    ADD com_file_change=p_dummy     The core code for open or new
    ADD com_exit                    Message sent here on exit accelerator
    CONSTANTS
        {
        O_COM_SYS_LAST          O_COM_EXIT
        BREAK_LINE_FOLLOWS      0x80        item is underlined
        PURE_COM_ID             0x7f        mask to exclude underline flag
        }
    }
```

The Series 3 version does not contain the defined constants BREAK_LINE_FOLLOWS and PURE_COM_ID.

Property

There is no property associated with the COMMAN class.

COMMAN methods

COM_INIT — Initialise

```
VOID com_init(VOID);
```

Perform application-specific initialisation. The supplied method does nothing.

A subclass may use this method to create and initialise one or more command manager component objects, such as a link paste server to implement the supply of data for the Bring command executed in another process.

The method is called from the WSERV `ao_init` method, immediately before WSERV receives a `WS_DYN_INIT` message. Any failure should result in `p_leave` being called: this will cause the start-up of the application to be aborted. In general, this method would not be expected to fail since the application's start-up heap should be calibrated to provide sufficient memory.

COM_STATWIN — Toggle permanent status window

```
VOID com_statwin(VOID);
```

This method is called when the application receives a Control-Menu keypress in circumstances explained in the description of the WSERV `ao_run` method. The supplied method does nothing.

A subclass may, if appropriate, keep a record of the status window visibility in its property and use this method to toggle the visibility of a permanent status window by appropriate calls to either `wsEnable` or `wsDisable` - and adjust the size of its display accordingly.

An application that can not reasonably alter its display size may choose not to subclass this method.

COM_ACCL_CHECK — Validate a command

```
INT com_accl_check(INT comid);
```

When an application command is selected, by either by pressing an accelerator key combination or by selecting a pull-down menu item, the command manager is sent the appropriate message, with message number `comid`. Before this message is sent, the command manager receives a `COM_ACCL_CHECK` message from the application's instance of WSERV (on the Series 3, this message is sent from the `ao_run` method and on the Series 3a, it is from the `ws_process_key` method, called from the `ao_run` method). If the `com_accl_check` method returns `FALSE`, the message with message number `comid` is not sent.

This method may therefore be used to check if the application is in a state to respond to the `comid` command. The Word application, for example, uses this method to ignore the PSION-+ accelerator if the Spell check software has not been installed.

The supplied method simply returns `TRUE`, allowing all command messages to be received at all times.

COM_MENU — A pull-down menu is about to appear

```
VOID com_menu(INT menu_num, PR_VAROOT *array);
```

This method is called when pull-down menu number `menu_num` (the leftmost menu is menu number zero) is about to appear. The supplied method does nothing.

At the time this method is called, the text of the menu has been loaded from a resource file, with the items held in successive elements of the variable array with handle `array`, but the menu window has not yet been made visible.

The data of the array is held in allocated memory. Each array element consists of a length byte followed by a `MENU_ITEM` struct, defined in *pulldown.g* as:

```
typedef struct
    {
    UBYTE com_id;    /* command manager method number */
    TEXT mn_txt[1]; /* zero terminated text string starts here */
    } MENU_ITEM;
```

and containing the command manager method function number, followed by the text, stored as a zero terminated string. As for the menus themselves, the first menu item is item zero, the second is item one, and so on.

A subclass may use the com_menu method to modify the menu contents according to the current state of the application, either to replace the text of one or more items - for example, to replace the text of the *second* item in the *third* menu:

```
TEXT *p;

if (menu_num==2)
    {
    p=(TEXT *)p_send3(array,O_VA_PREC,1); /* locate second item */
    hLoadResBuf(REPLACEMENT_TEXT_RID,p+2); /* skip byte count and method number */
    }
```

or to completely remove one or more items - for example, to remove the third item in the fourth menu:

```
if (menu_num==3)
    p_send3(array,O_VA_DELETE,2); /* delete third item */
```

These operations must be performed each time the menu bar is about to appear, since it is always reloaded from the resource file.

When replacing text note that, since the data is held in an allocated heap cell, any replacement text must not be longer than the original text for that item. The leading byte count allows system code to determine the amount of memory occupied by a menu item, even if the text has been replaced by text of a different length.

COM_MODE_CHANGE — Process a W_KEY_MODE key

```
VOID com_mode_change(INT shifted);
```

When WSERV receives an event of type WM_KEY with a keycode of W_KEY_MODE, in circumstances explained in the description of the WSERV ao_run method, it sends the command manager a COM_MODE_CHANGE message.

On receipt of this message the application should either do nothing or make some appropriate alteration to its state. On the Series 3 the Word application, for example, toggles in or out of Outline mode, the Data application switches in and out of data entry mode, while the Agenda application cycles around its views.

On the Series 3a the standard action is to cycle around the application's 'diamond list'.

If shifted is TRUE, in an application that cycles between three or more states, the direction of cycling should be reversed. Note that the shifted parameter should be ignored by code running on the Series 3.

The supplied method does nothing.

COM_FILE_CHANGE — Open or create a file

```
INT com_file_change(INT command,TEXT *pname);
```

On receipt of this message the command manager of a file-based application should close its current file, saving it if necessary and, depending on whether command is H_COMMAND_OPEN_FILE or H_COMMAND_CREATE_FILE, open or create the file whose file specification is pointed to by pname. Note that pname points to a transient cell and thus should not be passed as an argument to the am_new_filename method.

This message is sent to the command manager by WSERV as a result of the application receiving a Switchfiles message from the System Screen. The application itself may also send the command manager this message, for example, during start-up initialisation (normally in its ws_dyn_init method) and/or to implement the appropriate parts of its *New file* or *Open file* commands.

An application that is not file-based will not receive this message from WSERV.

The method should return zero to indicate that p_leave has not been called.

The supplied method does nothing other than to return zero.

See the description of the SHUTTER class in the chapter *The WSERV Class* for an explanation of the full processing of a Switchfiles message.

COM_EXIT — Exit the application

```
VOID com_exit(VOID);
```

This message may be received either as the result of the application receiving a Shutdown message from the System Screen or as the result of the user having selected the application's *Exit* command.

A file-based application must, if its file has changed, save the changes before exiting. If this fails it must come to the foreground and inform the user of the error, offering the user another chance to save the file.

The supplied method simply calls `p_exit(0)`. This is the recommended way to terminate the application, leaving the operating system to recover its resources.

CHAPTER 5

WINDOWS

A window is a rectangle in screen coordinates that provides a coordinate system for clipped drawing. Everything that is displayed by an HWIM application is drawn within one or more windows.

All HWIM windows subclass the WIN class, which is thus the ultimate superclass of all windows. The window classes supplied by HWIM are abstract classes and will normally need to be subclassed in order to create useful window objects.

The supplied classes, listed below, are described in the following sections of this chapter.

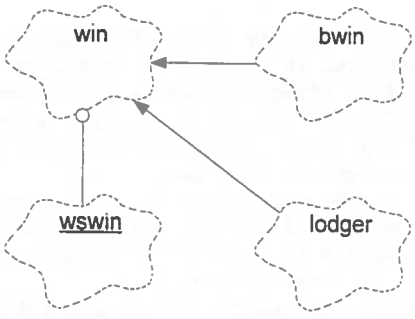
WIN	the ultimate window superclass.
BWIN	a subclass of WIN that draws a standard border around the window.
LODGER	a pseudo-window that subclasses WIN. Such a window is assumed to occupy a rectangular region within another window. The enclosing window (referred to as the *landlord*) will normally delegate all processing for that rectangle to the lodger.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the window concepts described in the *Introduction* and *Windows* chapters of the *Window Server Reference* manual
- the WSERV class, especially the `ao_run` method, which may send messages to a window
- graphics contexts and drawing, described in the *Graphics Output* chapter of the *Window Server Reference* manual

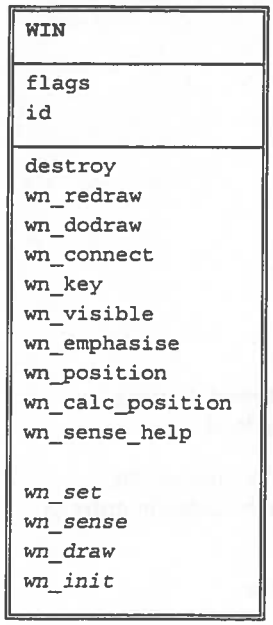
Class diagram



In addition to the methods and property in the application's code and data segments, a window has an associated data structure in the window server's resources, created by a call to `wCreateWindow` when an instance of WIN is itself created. The window server provides a set of functions that operate on such data structures, each data structure being uniquely identified by a window ID returned by `wCreateWindow`.

Since a uniquely identifiable data structure with a set of functions that operate on it is effectively an object, the window server resources associated with a window can be considered as a component object. The existence of the window server resources is thus indicated in the above class diagram by the 'using' relationship between WIN and a notional WSWIN (Window Server WINdow) class.

The WIN class



The WIN abstract class subclasses ROOT to form the superclass for all windows in HWIM applications.

As described earlier, a window effectively has a component window server object. However, as can be seen from the PROPERTY section of the following class definition, the WIN class does not store an object handle for this 'component'. Instead, it is referenced by means of a unique ID that is initialised in a call to a window's wn_connect method.

In effect, the wn_connect method performs the function of creating and initialising the window server 'component', in the same way that a normal component object is created and initialised in an object's initialisation method. It is therefore clear that the sending of a WN_CONNECT message is an essential part of the initialisation of any window (with the exception of 'lodger' windows, which are described later in this chapter).

Class definition

Defined in sub-category file *win.cl* (generated header file *win.g*).

```
CLASS    win      root
The window object superclass
    {
    REPLACE destroy          Close server window then supersend destroy
    ADD wn_redraw            Called in reponse to WM_REDRAW
    ADD wn_dodraw            Same effect as redraw but called by application
    ADD wn_connect           Connect the window to window server data (wswin)
    ADD wn_position          Calculate position and re-position
    ADD wn_calc_position     Calculate position
    ADD wn_key=p_false       For processing WM_KEY from server
    ADD wn_visible           Alters visibility state of window
    ADD wn_emphasise         Window is highlighted in some way
    ADD wn_sense_help        Give start ID for help
    DEFER wn_set             Set some window object property fields
    DEFER wn_sense           Sense some window object property fields
    DEFER wn_draw            Draw to existing graphics context - usually subclassed
    DEFER wn_init            Specific initialisation for a type of window
```

```
CONSTANTS
    {
    CURSOR_WIDTH        2
    CURSOR_COLWID       2
    SYSTEM_FONT_COLWID  2
    W_KEY_SPACE         32
    PR_BWIN_CUSHION     0x1     matches W_BORD_CUSHION
    PR_BWIN_CORNER_4    0x2     matches W_BORD_CORNER_4
    PR_BWIN_SHADOW_1    0x4     matches W_BORD_SHADOW_S
    PR_BWIN_SHADOW_2    0x8     matches W_BORD_SHADOW_D
    PR_WIN_EMPHASISED   0x10    matches W_BORD_SHADOW_ON
    PR_BWIN_OPEN        0x20    matches W_BORD_OPEN
    PR_BWIN_CORNER_1    0x40    matches W_BORD_CORNER_1
    ! Gap of four bits, reserved for the arrow bits, for subclasses of BWIN only
    ! A dialog control is always a LODGER subclass, so can use 1 of the 4 bits
    PR_WIN_IS_DLCTRL    0x80    set by LODGER subclass whose landlord is a DLG
    PR_WIN_INITIALISED  0x800   Window completely initialised
    PR_WIN_FORCE_RIGHT  0x1000
    PR_WIN_FORCE_LEFT   0x2000
    PR_WIN_FORCE_BOTTOM 0x4000
    PR_WIN_FORCE_TOP    0x8000
    PR_WIN_FORCE_FLAGS  0xf000
    WIN_3dBORDER        PR_WIN_FORCE_RIGHT  Alternative use, for S3a border style
    WIN_FROM_ATS        PR_WIN_FORCE_LEFT   Alternative use, for auto test system
    IN_WIN_EMPHASISED   (PR_WIN_EMPHASISED)

    WV_INVISIBLE        0
    WV_VISIBLE          1
    WV_INITINVIS        2
    WV_INITVIS          3

    WN_KEY_NO_CHANGE    0
    WN_KEY_CHANGED      3
    WN_KEY_CHANGED_DEFER 7
    WN_KEY_CANCELLED    (-1)
    WN_KEY_ABSORB_ON    (-2)

    ERROR_RID_OFFSET    512
    }

PROPERTY
    {
    UWORD id;           window server window ID
    UWORD flags;        PR_WIN EMPHASISED etc.
    }
}
```

The Series 3 and Series 3a versions of the class definition do not contain the defined constant `PR_WIN_IS_DLCTRL`.

The Series 3 version of the class definition does not contain the defined constants `WIN_3dBORDER` and `WIN_FROM_ATS`.

It does, however, contain the following additional defined constants:

```
SCREEN_WIDTH            240
SCREEN_HEIGHT           80
CURSOR_HEIGHT           10
SYSTEM_FONT_HEIGHT      8
SYSTEM_FONT_ASCENT      7
SYSTEM_FONT_NUM_WIDTH   6
CONTROL_HEIGHT          (SYSTEM_FONT_HEIGHT+2)
SYSTEM_FONT_ID          WS_FONT_BASE
BOLD_FONT_ID            WS_FONT_BASE+1
DIGITS_FONT_ID          WS_FONT_BASE+2
SYSTEM_MONO_WIDTH       6
BOLD_MONO_WIDTH         8
DEFAULT_ICON_ID         WS_BITMAP_BASE+2
```

On the Series 3a the corresponding values are read or derived from the globally accessible data that is described in the *Introduction* chapter of this manual.

Property

`win.id`	the window ID, returned from a call to `wCreateWindow`, and used to identify the window server data structures associated with the window
`win.flags`	a collection of flags recording the state of the window, as described below

Window flags

The content of `win.flags` may be any legal combination of the following flags:

`PR_WIN_EMPHASISED`	`TRUE` if the window is to be drawn in a highlighted state, the flag is set or cleared by the `wn_emphasise` method
`PR_WIN_INITIALISED`	`TRUE` if some significant aspect of the window initialisation is successfully completed, normally implying some change in action during subsequent destruction of the window. This is currently used by only the `DLGBOX` subclass
`PR_WIN_IS_DLCTRL`	set `TRUE` , by `DLGBOX` methods, for a `LODGER` subclass if its landlord window is an instance of (a subclass of) `DLGBOX`. Introduced for the Work*about* to enable dialog controls to determine if they need to draw themselves in a small font.

Additional flag bits are used by the `BWIN` subclass.

The following flag values are used by system code, for example by the `wn_calc_position` method, during the initialisation of a number of types of window, but they are never set in `win.flags`:

`PR_WIN_FORCE_RIGHT`	the window is to be positioned at the right hand edge of the screen. This flag is mutually exclusive with `PR_WIN_FORCE_LEFT`
`PR_WIN_FORCE_LEFT`	the window is to be positioned at the left hand edge of the screen. This flag is mutually exclusive with `PR_WIN_FORCE_RIGHT`
`PR_WIN_FORCE_BOTTOM`	the window is to be positioned at the bottom edge of the screen. This flag is mutually exclusive with `PR_WIN_FORCE_TOP`
`PR_WIN_FORCE_TOP`	the window is to be positioned at the top edge of the screen. This flag is mutually exclusive with `PR_WIN_FORCE_BOTTOM`

WIN methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the window, including its window server data structure, provided this has previously been successfully created. An application will not normally send this message to its client window (or to any permanent component of the client window).

The method ensures that no window server event can be directed to the window after its destruction sequence is initiated, and calls `wCloseWindowTree`, passing `win.id`, to free the window's window server data.

Finally, the method supersends the `DESTROY` message, to destroy the window's client-side resources (including automated destruction of any component objects).

WN_CONNECT Create the window's window server data

```
VOID wn_connect(PR_WIN *par, UINT fields, W_WINDATA *windata);
```

Create the window server data structure for the window by calling the window server function `wCreateWindow`, writing the returned ID to `win.id`, as indicated by the following code:

```
METHOD VOID win_wn_connect(PR_WIN *self,PR_WIN *par,UINT fields,W_WINDATA *wdata)
    {
    self->win.id=wCreateWindow((par?par->win.id:0),fields,wdata,(UWORD)self);
    }
```

The parameters `windata` and `fields` are as described for `wCreateWindow` in the *Windows* chapter of the *Window Server Reference* manual. The value of `par` should be `NULL` if the window is to be a top-level window, otherwise it should be the handle of the parent window in the window tree. The value of `self` is passed to `wCreateWindow` for use by the window server as a handle to identify the window, for example, when the window must be sent a `WN_REDRAW` message. The window server uses `self` in much the same way that window method functions use `win.id`.

In general, HWIM windows are created with the `W_WIN_BACK_BITMAP` bit in `windata->flags` cleared, so that they will be redrawn by the window server `WM_REDRAW` mechanism, which causes the window to be sent a `WN_REDRAW` message.

Assuming that the window server function `wDisableLeaves` has not been called, any failure will result in `p_leave` being called.

This method must be executed during the initialisation of any window (other than 'lodger' windows - see later).

Application code will frequently replace this method to customise the window's features and/or to create child windows.

WN_REDRAW System-initiated redraw

```
VOID wn_redraw(P_RECT *prect);
```

This method is intended to be called by system code to redraw the window, following receipt by `WSERV` of a window server event of type `WM_REDRAW`. An application will not normally replace this method, nor will it explicitly send a `WN_REDRAW` message to any of its windows.

The operation of the method is as illustrated in the following code:

```
VOID win_wn_redraw(PR_WIN *self, P_RECT *prect)
    {
    wBeginRedrawWinGC0(self->win.id);
    p_send2(self,O_WN_DRAW);
    wEndRedraw();
    }
```

Note that the whole window is redrawn, the rectangle coordinates pointed to by `prect` being ignored. A window that selectively redraws only the area specified by `prect` should replace this method.

WN_DODRAW Application-initiated redraw

```
VOID wn_dodraw(VOID);
```

This method is intended to be called by the application itself, for example, following a change in the data being displayed. An application will not normally subclass this method.

The operation is illustrated in the following code:

```
VOID win_wn_dodraw(PR_WIN *self)
    {
    wValidateWin(self->win.id);
    gCreateTempGC0(self->win.id);
    p_send2(self,O_WN_DRAW);
    gFreeTempGC();
    wFlush();
    }
```

The use of this method is equivalent to making a call to the window server function `wInvalidateWin`, except that the redrawing is performed immediately, without having to wait for a redraw event to arrive from the window server.

WN_VISIBLE — Set visibility

```
INT wn_visible(UINT flag);
```

Set the visibility of the window, and any descendant windows, according to the value of `flag`. An application will not normally subclass this method.

During initialisation of a window, possible values of flag are:

`WV_INITVIS` which calls `wInitialiseWindowTree(win.id)`

`WV_INITINVIS` which calls `wMakeInvisible(win.id)` and then `wInitialiseWindowTree(win.id)`.

For an existing initialised window, this method may be called with flag set to either `WV_VISIBLE` or `WV_INVISIBLE` which respectively call `wMakeVisible(win.id)` or `wMakeInvisible(win.id)`.

This method will normally be called during the initialisation of a window. See also the HWIM utility function `hInitVis`.

The method returns zero and is thus suitable for being called under the protection of `p_enter`.

WN_EMPHASISE — Set window highlight

```
VOID wn_emphasise(UINT flag);
```

Set or clear the `PR_WIN_EMPHASISED` bit in `win.flags`, depending on whether flag is `TRUE` or `FALSE`. Then call `wInvalidateWin(win.id)` so that the window will eventually receive a `WN_REDRAW` message.

Windows generally draw themselves differently in some way, according to whether or not they are emphasised.

This method is not suitable for use in quality applications. It is expected that a subclass will always replace this method to provide specific and more effective redraw logic (see, for example the `BWIN` subclass).

WN_SENSE_HELP — Sense start ID for help

```
INT wn_sense_help(VOID);
```

Sense the resource ID for the application's current Help index.

Returns the value of `w_ws->wserv.help_index_id` or, if this value is zero, the (negative) system resource ID `-SYS_HELP_ON_HELP`.

This method can be subclassed to facilitate context-sensitive Help.

WN_CALC_POSITION — Calculate a window position

```
VOID wn_calc_position(INT flags, P_EXTENT *pext);
```

Calculate, and write to `pext->tl.x` and `pext->tl.y`, the required screen coordinates of the top left corner of the window of width `pext->width` and height `pext->height` to place it on the screen in the position specified by `flags`. The calculation assumes that the window includes a surrounding blank 'cushion', one pixel wide.

The value of flags may contain any `OR`ed combination of:

- either `PR_WIN_FORCE_RIGHT` or `PR_WIN_FORCE_LEFT`, with obvious meanings. In either case the window is positioned so that the single pixel cushion at the side of the window that touches the edge of the screen is not visible. If neither flag is present the window is centred horizontally
- either `PR_WIN_FORCE_TOP` or `PR_WIN_FORCE_BOTTOM`, again with obvious meanings. In either case the window is positioned so that the single pixel cushion at the side of the window that touches the edge of the screen is not visible. If neither flag is present the window is centred vertically

An application will not normally need to either replace or make explicit calls to this method.

WN_POSITION — Set window position

```
VOID wn_position(INT flags);
```

Position the window according to the value of flags, which may contain any ORed combination of:

- either PR_WIN_FORCE_RIGHT or PR_WIN_FORCE_LEFT, with obvious meanings. In either case the window is positioned so that the single pixel cushion at the side of the window that touches the edge of the screen is not visible. If neither flag is present the window is centred horizontally
- either PR_WIN_FORCE_TOP or PR_WIN_FORCE_BOTTOM, again with obvious meanings. In either case the window is positioned so that the single pixel cushion at the side of the window that touches the edge of the screen is not visible. If neither flag is present the window is centred vertically

The method sends a WN_CALC_POSITION message to determine the required position before moving the window by means of a call to wSetWindow.

If used, this method will normally be called during the initialisation of a window, but must not be called before the window has processed a WN_CONNECT message.

An application will not normally need to replace this method

WN_KEY — Process a keypress

```
INT wn_key(INT keycode, INT modifiers);
```

Process a keypress, where keycode is the code of the key pressed and modifiers contains a set of flags indicatinf which modifier keys (SHIFT, CTRL etc.) were held down when the key was pressed. The possible values of keycode and modifiers are described in the *Events* chapter of the *Window Server Reference* manual.

Depending on the type of keypress and the current state of the application, the WN_KEY message may be sent to a window (normally the client window or a dialog) by system code - notably the ws_process_key method of the application's instance of (a subclass of) WSERV.

The supplied method does nothing other than return zero (WN_KEY_NO_CHANGE).

A subclass will, in general, replace this method. In many cases the wn_key method of a window may delegate the processing by sending WN_KEY messages to one or more other windows.

Note that the return value generally only has significance for subclasses of WIN. A dialog, for instance, will return a non-zero value to indicate that the keypress should terminate the dialog and a menu bar may return the ID of a command menu option to be executed. See the description of the WSERV ws_process_key method for the way in which system code interprets wn_key return values from various types of window.

The possible return values given by the various WN_KEY_XXX constants specified in the WIN class definition are of particular significance for the wn_key methods of the DLGBOX class and the dialog component classes. See the *Dialog Boxes* chapter and the following dialog box component chapters for further details.

Deferred WIN methods

A window will generally replace one or more of the methods described below. Note that there is no requirement for any particular subclass to replace all these methods.

WN_INIT — Initialise

```
VOID wn_init(...);
```

Provide class-specific initialisation. For most windows this will include a call of the form:

```
p_send5(self,O_WN_CONNECT,parent,ws_flags,&ws_data);
```

The number and types of parameters to the wn_init method depend on the class. However, once specified for a particular class, further subclasses will normally follow the same parameter structure.

Note that a simple window may not need to supply this method, since all essential initialisation may be performed by the `wn_connect` and `wn_visible` methods.

WN_SET — Set property

```
VOID wn_set(...);
```

Set one or more property fields.

The number and types of parameters to the `wn_set` method depend on the class. However, once specified for a particular class, further subclasses will normally follow the same parameter structure.

WN_SENSE — Sense property

```
VOID wn_sense(...);
```

Sense one or more property fields.

The number and types of parameters to the `wn_sense` method depend on the class. However, once specified for a particular class, further subclasses will normally follow the same parameter structure.

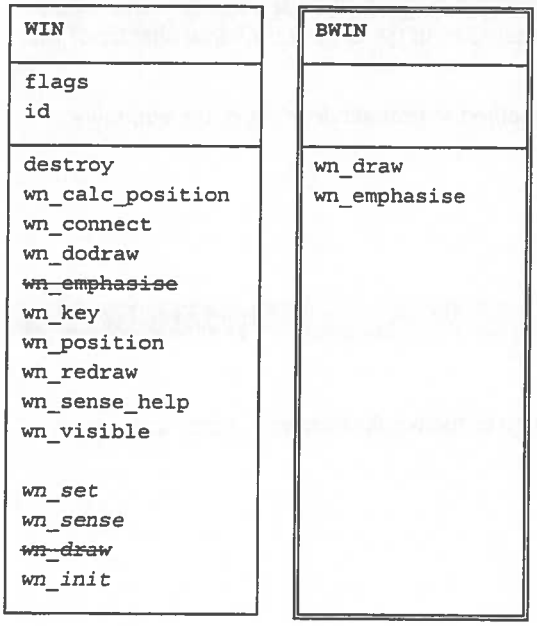
WN_DRAW — Draw to existing GC

```
VOID wn_draw(VOID);
```

Supply class-specific drawing for the whole area of the window, called from the `wn_redraw` and `wn_dodraw` methods. Most application subclasses will need to supply this method.

The method assumes that an appropriate graphics context exists, so the caller is responsible for supplying a graphics context. Note, however that this method is called from the `wn_redraw` and `wn_dodraw` methods, both of which create a basic graphics context around the sending of a `WN_DRAW` message.

The BWIN bordered window class



The BWIN abstract class is the superclass for all bordered windows.

Class definition

Defined in sub-category file *bwin.cl* (generated header file *bwin.g*).

```
CLASS   bwin      win
All windows that have a border use this
    {
    REPLACE wn_draw             Redraw border
    REPLACE wn_emphasise        Update border
    CONSTANTS
        {
        IN_BWIN_CORNER_4        PR_BWIN_CORNER_4
        IN_BWIN_SHADOW_1        PR_BWIN_SHADOW_1
        IN_BWIN_SHADOW_2        PR_BWIN_SHADOW_2
        IN_BWIN_CUSHION         PR_BWIN_CUSHION
        IN_BWIN_OPEN            PR_BWIN_OPEN
        BWIN_CUSHION_X          1
        BWIN_CUSHION_Y          1
        BWIN_SHADOW_1_HEIGHT    1
        BWIN_SHADOW_1_WIDTH     1
        BWIN_SHADOW_2_HEIGHT    2
        BWIN_SHADOW_2_WIDTH     2
        }
    }
```

Property

None.

BWIN methods

WN_DRAW — Draw border

```
VOID wn_draw(VOID);
```

Draw the window's border by calling

```
gBorder(self->win.flags);
```

The significant flag bits are PR_BWIN_CUSHION to PR_BWIN_CORNER_1 inclusive, together with the following four bits (0x80 to 0x400 inclusive) used to indicate the corner arrows. These bits are equivalent to the window server flags W_BORD_CUSHION to W_BORD_CORNER_1 and W_BORD_TOP_ON to W_BORD_BOT_OFF, whose effects are explained in the description of gBorder and gBorderRect in the *Graphics Output* chapter of the *Window Server Reference* manual.

An application-specific subclass will normally replace this method to provide drawing of the window's content, including the line:

```
p_supersend2(self,O_WN_DRAW);
```

to draw the border.

WN_EMPHASISE — Update border

```
VOID wn_emphasise(UINT flag);
```

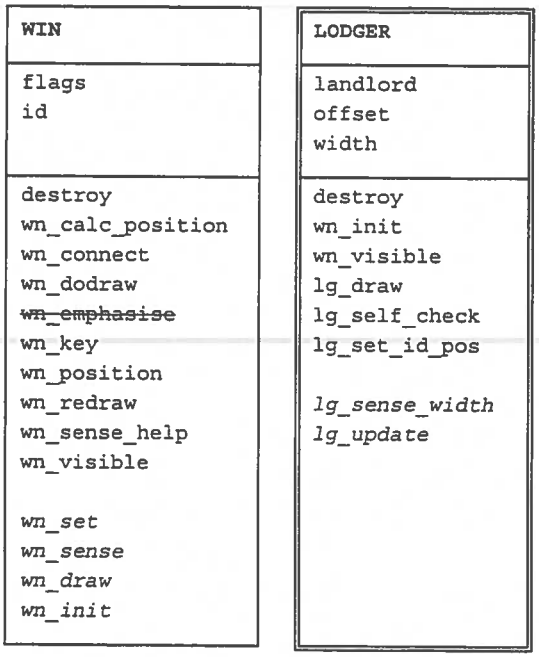
If flag is FALSE, clear the PR_WIN_EMPHASISED flag (equivalent to the window server W_BORD_SHADOW_ON flag) in win.flags, otherwise set it.

Then redraw the border as indicated in the following code:

```
gCreateTempGC0(self->win.id);
gBorder(self->win.flags);
gFreeTempGC;
```

The shadow of a shadowed bordered window is only visible when the window is emphasised.

The LODGER class



The LODGER class defines a window that does not have its own independent window server data structure, and therefore does not have an independent window server ID. A *lodger window* is defined to be any window that subclasses LODGER.

A lodger window occupies a rectangular region within another window, referred to as the lodger window's *landlord*, and shares the window ID of the landlord window. In other respects a lodger window has broadly similar behaviour to a normal window, supporting a similar set of drawing and key processing messages. A lodger window can be considered to take over the responsibility for drawing the content of a rectangular region of the landlord window.

A significant difference between the two types of window is that drawing in a lodger window is not clipped by the boundaries of the lodger window itself. Drawing will only be clipped by the landlord window rectangle. In consequence, a lodger window has a duty never to draw outside its bounding rectangle.

The most common single use for a lodger window is as a component control within a dialog box.

Since a lodger window does not have an independent window server data structure, it is more efficient in terms of memory usage. This can be significant in the case of a complex compound window (a dialog box, for example, can easily need ten or more component sub-windows). An additional advantage is that such a compound window will scroll more smoothly - the scroll mechanism operates on only the single 'real' window and does not require messages to be sent to any of the component lodger windows (except for those that have freshly exposed regions to draw).

Subclasses of LODGER (and, indeed, any windows that do not subclass BWIN) are free to reuse the flag bits PR_BWIN_CUSHION to PR_BWIN_CORNER_1 inclusive, together with the four flag bits 0x80 to 0x400 inclusive that are used to indicate corner arrows.

Class definition

Defined in sub-category file *lodger.cl* (generated header file *lodger.g*).

```
CLASS lodger    win
Lodger window
    {
    REPLACE destroy=root_destroy
    REPLACE wn_init              Set up landlord
    REPLACE wn_visible
    ADD lg_set_id_pos            Set window ID after dynamic initialisation
    ADD lg_draw                  Temp GC and call wn_draw
    ADD lg_self_check=p_true     Returns TRUE if contents are legal
    DEFER lg_sense_width         For dialogs to set their sizes
    DEFER lg_update              For fnselwn and fnedit

    CONSTANTS
        {
        LG_CHECK_OK              1     Lodger self checked o.k., no change
        LG_CHECK_FAILED          0     Lodger failed self check, no change
        LG_CHECK_FAILED_CHANGED (-1)   Lodger failed self check, changed
        LG_CHECK_OK_CHANGED     (-2)   Lodger self checked ok., changed
        }

    PROPERTY
        {
        P_POINT offset;        offset into landlord window
        UWORD width;           width available to control
        PR_WIN *landlord;      owner window
        }
    }
```

Property

`lodger.offset`	the pixel coordinates of the top left corner of the lodger window with respect to the top left corner of the landlord window
`lodger.width`	the width, in pixels, of the area available to the lodger window
`lodger.landlord`	the handle of the window's landlord window, assumed to be (a subclass of) WIN. The lodger window has access to the window server ID of the 'real' window (the landlord) via `lodger.landlord->win.id`

A lodger window's property does not record the height of the lodger window. This is suitable when a lodger window is a component of a dialog box since, in this case, it always has a fixed height of CONTROL_HEIGHT pixels.

LODGER methods

WN_INIT — Initialise

```
VOID wn_init(UINT *par, PR_WIN *landlord);
```

Initialise the lodger window by copying the value of `landlord`, which should be the handle of the owning landlord window, into `lodger.landlord`.

The method ignores the value of `par`, which is specified for use by subclassers.

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy only the client-side resources of the lodger window, together with automatic destruction of any components.

Calls the `ROOT` `destroy` method directly, to avoid calling the `destroy` method at the `WIN` level, which might free the landlord's window server data structure.

WN_VISIBLE — Set visibility

```
VOID wn_visible(UINT flag);
```

Make the lodger window invisible if `flag` is `FALSE`, otherwise make the lodger window visible. (Note that, in contrast to the `WIN` superclass, only two flag values are supported.)

If the lodger window is being made visible, the method copies the landlord window's window server ID into `win.id` and sends an `LG_DRAW` message to draw the lodger window's contents.

If the lodger window is being made invisible, `win.id` is set to zero and the method creates a temporary graphics context, calls `gClrRect` to clear the area in the landlord window that is occupied by the lodger window and then frees the temporary graphics context.

A non-zero value of `win.id` thus indicates that the window is visible and this test is used, for example, by the `lg_draw` method. Setting `win.id` to be a copy of the value in `lodger.landlord->win.id` is an implementation decision that simplifies access to the window server ID of the 'real' window.

LG_SET_ID_POS — Set ID, position and width

```
VOID lg_set_id_pos(INT id, P_POINT *pos, UINT width);
```

Set `win.id` to the value of `id`, copy `*pos` into `lodger.offset` and set `lodger.width` to `width`.

A typical call to this method will be from a dialog box after all the items have been loaded and the dialog box has been set to the required width to display all its items.

LG_DRAW — Create GC and draw

```
VOID lg_draw(VOID);
```

If `win.id` is non-zero, indicating that the window is in the visible state, draw the window content:

```
gCreateTempGC0(self->win.id);
p_send2(self,O_WN_DRAW);
gFreeTempGC();
```

Otherwise do nothing.

This method is intended to be called following a change in the data displayed in the lodger window (for example, at the conclusion of a lodger window's `wn_set` method). Since it creates its own temporary graphics context, it must never be called when a temporary graphics context exists, for example, as a result of the landlord window receiving a `WN_REDRAW` message.

LG_SELF_CHECK — Check content is valid

```
INT lg_self_check(INT item, INT can_defer);
```

Perform a check that any property contains valid data. A numeric editor may, for example, check that its current value is within its allowed upper and lower bounds.

Possible return values are as follows:

LG_CHECK_OK	the data has not changed and the check succeeded
LG_CHECK_FAILED	the data has not changed and the check failed
LG_CHECK_FAILED_CHANGED	the data has changed and the check failed
LG_CHECK_OK_CHANGED	the data has changed and the check succeeded

The supplied method just returns LG_CHECK_OK.

Deferred LODGER methods

These methods are intended to be replaced by classes that are used as components of a dialog box.

LG_SENSE_WIDTH — Sense required width

```
INT lg_sense_width(VOID);
```

This method is expected to return the width, in pixels, required to draw the lodger window contents.

An object will only receive this message when it is a component of a dialog box (see the DLGBOX dl_set_size method, described in the *Dialog Boxes* chapter).

Some system-supplied subclasses assume that this method is called only once in the lifetime of an instance.

LG_UPDATE — Update file name

```
VOID lg_update(TEXT *pack, INT derr);
```

This method is defined for the FNSELWN and FNEDIT dialog box component classes. See the descriptions of these classes for an explanation of the method.

CHAPTER 6

LIST BOXES AND MENUS

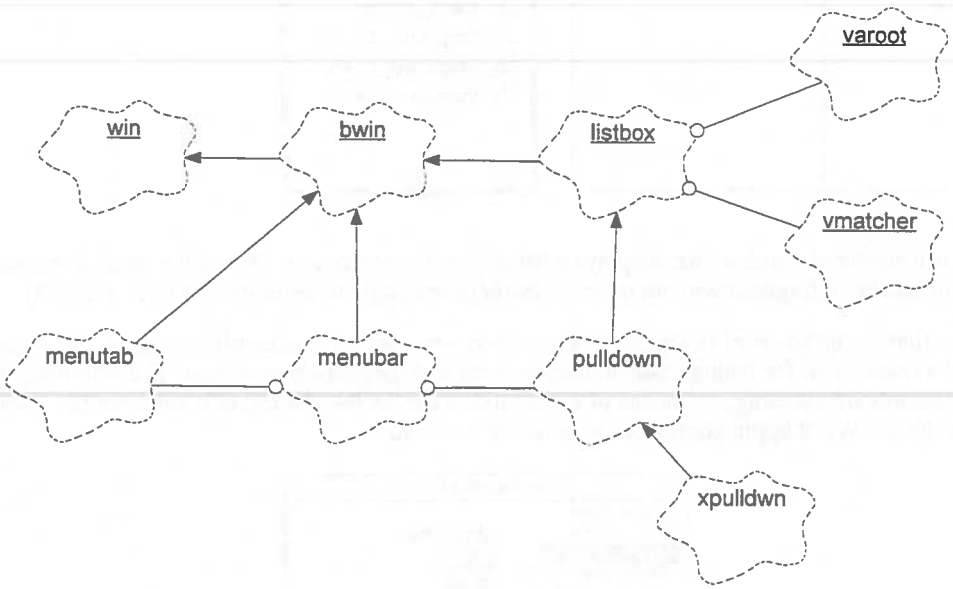
All HWIM applications are expected to use commands and will normally require a menu bar and its associated pull-down menu(s). A menu bar is implemented with the aid of the LISTBOX class, which forms a basic component of a number of other HWIM window objects.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the descriptions of the WIN and BWIN classes in the *Windows* chapter
- the creation of a menu bar and the sending of WN_KEY messages to a menu bar from the WSERV ao_run method, as described in the chapter *The WSERV class*

Class diagram



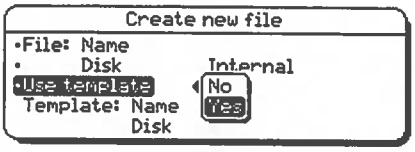
List boxes

WIN	BWIN
flags id	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help wn_visible *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ ~~wn_emphasise~~

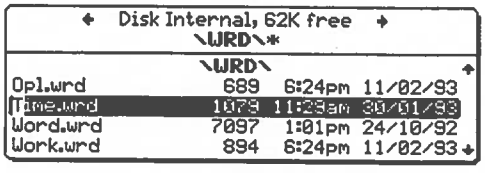
LISTBOX
match va flags width matchlen curoff offset current top first last vastart matchstart
destroy wn_init wn_key wn_draw wn_emphasise lb_draw_item lb_draw_emphasis lb_size_window lb_item_width lb_take_focus lb_inquire_focus lb_inquire_item lb_inquire_last

A list box is a bordered window that displays a list of items, one per line. One of the items is normally the *current* item and is highlighted with an inverse *obloid* (a rectangle without its four corner pixels).

List boxes - that is, instances of (a subclass of) LISTBOX - are used, for example, to display the expanded contents of a *choice list*, for listing items in *help screens* and for *pull-down menus*. The following diagram shows an example of selecting, by means of an expanded choice list, the use of a template file - within the dialog used by the Word application's *Create new file* command.



List boxes are also used to display *file lists*, as shown in the following diagram. This illustrates that a list box may have a title consisting of one or more lines, separated from the main list by a horizontal line. The main list may contain more items than can be shown on the screen. The presence of further items before or after those that are visible can be indicated by up and down arrows in a right hand scroll gutter.



The current item may be changed by means of the page up, page down and cursor keys. If there are more items than can be displayed in the list box window, the contents will be scrolled so that the current item remains visible.

The LISTBOX class provides support for *incremental matching*, whereby successive keypresses select the first item in the list box that starts with the sequence of matching characters that has so far been typed in. (This behaviour is exhibited, for example, by the file selector, displayed by selecting the file name item in the dialog for any file-based command and pressing the Tab key.)

The items displayed in a list box are assumed to be stored in a variable array (an instance of a subclass of the OLIB VAROOT class). The data in each individual element of this array is assumed to be stored contiguously rather than, say, crossing the boundary between two or more allocated memory cells. Each element is assumed to contain a zero terminated string (which may, however, be preceded by a constant number of bytes of other data). There is no requirement for adjacent records to be contiguous with each other.

Class definition

Defined in sub-category file *listbox.cl* (generated header file *listbox.g*).

```
CLASS   listbox         bwin
List box
    {
    REPLACE destroy
    REPLACE wn_init
    REPLACE wn_key
    REPLACE wn_draw
    REPLACE wn_emphasise

    ADD lb_draw_item            Draw an item by index number
    ADD lb_draw_emphasis        Draw/undraw emphasis on an item
    ADD lb_size_window          Resize the window to the correct size
    ADD lb_item_width           Returns the required width of the item
    ADD lb_take_focus           Move focus item specified by index
    ADD lb_inquire_focus        Returns index of item with focus
    ADD lb_inquire_item         Get pointer to item text
    ADD lb_inquire_last         Get index of last item

    CONSTANTS
        {
        IN_LISTBOX_MATCHER      0x0001
        IN_LISTBOX_KEEP_ARRAY   0x0002
        IN_LISTBOX_TEXT_OFFSET  0x0004
        IN_LISTBOX_CUR_SET      0x0008
        IN_LISTBOX_FIRST_SET    0x0010
        IN_LISTBOX_LAST_SET     0x0020
        IN_LISTBOX_POS_ALIGN_X  0x0040
        IN_LISTBOX_POS_ALIGN_Y  0x0080
        IN_LISTBOX_MIN_WIDTH    0x0100
        IN_LISTBOX_AUTO_SIZE    0x0200
        IN_LISTBOX_FORCE_WIDE   0x0400
        IN_LISTBOX_WRAP_ROUND   0x0800
        IN_LISTBOX_FIXED_WIDTH  0x8000

        PR_LISTBOX_KEEP_ARRAY   IN_LISTBOX_KEEP_ARRAY
        PR_LISTBOX_FORCE_WIDE   IN_LISTBOX_FORCE_WIDE
        PR_LISTBOX_WRAP_ROUND   IN_LISTBOX_WRAP_ROUND
        PR_LISTBOX_UNSTABLE     0x1000
        PR_LISTBOX_BOLD_CURSOR  0x2000
        PR_LISTBOX_PLAQUE       0x4000
        PR_LISTBOX_FIXED_WIDTH  IN_LISTBOX_FIXED_WIDTH
        PR_LISTBOX_SMALL_FONT   PR_WIN_FORCE_RIGHT set into win.flags property

        LISTBOX_TOP_EDGE        (BWIN_CUSHION_Y+2)
        LISTBOX_BOTTOM_EDGE     (LISTBOX_TOP_EDGE+BWIN_SHADOW_1_HEIGHT)
        LISTBOX_LEFT_EDGE       (BWIN_CUSHION_X+2)
        LISTBOX_RIGHT_EDGE      (LISTBOX_LEFT_EDGE+BWIN_SHADOW_1_WIDTH)
        XLISTBOX_TOP_EDGE        (LISTBOX_TOP_EDGE+6)
        XLISTBOX_BOTTOM_EDGE     (LISTBOX_BOTTOM_EDGE+5)
        XLISTBOX_LEFT_EDGE       (LISTBOX_LEFT_EDGE+6)
        XLISTBOX_RIGHT_EDGE      (LISTBOX_RIGHT_EDGE+5)
        LISTBOX_OBLOID_INDENT    2  extra width on each side for highlight obloid
        LISTBOX_TITLELINE_HEIGHT 3  additional height for line below a title
        LISTBOX_SCROLL_GUTTER    7  horizontal space for a scroll indicator
        LISTBOX_SCROLL_OFFSET    1
        }
```

```
TYPES
    {
    typedef struct
        {
        UWORD flags;
        PR_VAROOT *array;
        UWORD offset;
        WORD current;
        UWORD top;
        UWORD first;
        UWORD last;
        P_POINT pos;
        UWORD minwid;
        } IN_LISTBOX;
    }

PROPERTY 1
    {
    PR_VMATCHER *match;     matcher object if any
    PR_VAROOT *va;          item list
    UWORD flags;            holds PR_LISTBOX flags
    WORD width;             Width of the list box in pixels
    UWORD matchlen;         for matcher
    UWORD curoff;           offset of text for matcher cursor
    UWORD offset;           offset of text in VA item
    WORD current;           current selection
    WORD top;               top visible item in scrolling section
    WORD first;             first item in scrolling section
    WORD last;              last item in scrolling section
    WORD vastart;           index in listbox where va starts
    WORD matchstart;        index in listbox where matching starts
    }
}
```

The Series 3 version does not contain the definitions of `IN_LISTBOX_FIXED_WIDTH`, `PR_LISTBOX_PLAQUE`, `PR_LISTBOX_FIXED_WIDTH`, `XLISTBOX_TOP_EDGE`, `XLISTBOX_BOTTOM_EDGE`, `XLISTBOX_LEFT_EDGE` and `XLISTBOX_RIGHT_EDGE`.

It does, however, contain an additional `CONSTANT` definition:

```
LISTBOX_ITEM_HEIGHT     (SYSTEM_FONT_HEIGHT+2)
```

On the Series 3a, this value is derived from the data described in the *Globally accessible data* section of the *Introduction* chapter.

The constant `PR_LISTBOX_SMALL_FONT` is introduced in the class definition for the Work*about*.

Property

`listbox.match`	Either `NULL` or the handle of an instance of `VMATCHER`. Incremental matching is enabled if this element is not `NULL`.
`listbox.va`	The handle of an instance of a subclass of `VAROOT` containing the items that are displayed in the list box. By default each item is assumed to contain zero terminated text, whose first character is at a byte offset of `listbox.offset` within the item.
`listbox.flags`	A combination of the state flags listed below.
`listbox.width`	The width, in pixels, of the region within the list box that is used to display items (not counting the window borders and any scroll gutter).
`listbox.matchlen`	Provided incremental matching is enabled, the number of characters that have currently been incrementally matched. The address of this item is passed to any `VMATCHER` component, which automatically updates its value.
`listbox.curoff`	Provided incremental matching is enabled, the current horizontal offset to the incremental matching cursor.

`listbox.offset`	The byte offset within each element of the `listbox.va` array to the first character of the text to be displayed.
`listbox.current`	The index number of the currently highlighted item.
`listbox.top`	The index of the first *visible* item in the main list; its value is always greater than or equal to `listbox.first`.
`listbox.first`	The index of the first item that forms part of the main list of items in the list box. If non-zero, items with indexes in the range 0 to `listbox.first-1` are drawn as the list box title, separated from the main list by a horizontal line.
`listbox.last`	If not zero, the index of the last item from the `listbox.va` array to be displayed in the main list. Setting `listbox.last` allows trailing array items to be excluded from the list.
`listbox.vastart`	The index of the first item whose content is stored in the `listbox.va` array, allowing subclasses (see, for example, the `FILELIST` class) to derive items, such as their title, from other sources. Such subclasses must replace the `lb_inquire_item` method.
`listbox.matchstart`	The index of the item at which incremental matching is to start - intended for use by the `FILELIST` subclass. No incremental matching cursor is ever drawn in an item whose index number is less than `listbox.matchstart`.

List box flags

The content of `listbox.flags` may be any combination of the following flags:

`PR_LISTBOX_KEEP_ARRAY`	If set, the `listbox.va` array is not sent a `DESTROY` message on destruction of the list box.
`PR_LISTBOX_FORCE_WIDE`	If set, the list box is drawn to include a scroll gutter, even if all the list items can be displayed within the list box window.
`PR_LISTBOX_WRAP_ROUND`	If set, pressing the up cursor key when positioned on the first item in the list, or the down cursor key when positioned on the last item in the list will wrap between the first and last items.
`PR_LISTBOX_UNSTABLE`	This flag should be set by a `LISTBOX` subclass to indicate that the contents of the listbox.va array are in an inconsistent state. It is used in this way by the `FILELIST` class.
`PR_LISTBOX_BOLD_CURSOR`	Should be set if all selectable list box items are displayed in bold text.
`PR_LISTBOX_PLAQUE`	If set, the listbox is drawn with a 3-dimensional style grey and black border, otherwise a simple black border is used, as on the Series 3. This flag is not supported on Series 3 machines.
`PR_LISTBOX_FIXED_WIDTH`	This flag should only be set on initialisation by including `IN_LISTBOX_FIXED_WIDTH` in the flags field of the `IN_LISTBOX` inintialisation struct. If set, the width of the listbox is exactly as specified by the minwid element of the `IN_LISTBOX` struct. Otherwise, the width of the listbox is adjusted if necessary to accommodate the widest item. This flag is not supported on Series 3 machines.

The contents of the remaining bit fields of `listbox.flags` are undefined.

On the Work*about*, an additional flag may be set in `win.flags`:

`PR_LISTBOX_SMALL_FONT`	On the Work*about*, this flag may be set in the `win.flags` property of the list box, to indicate that the list box contents should be displayed in the small font. This flag is currently only set on initialisation of the Work*about* `FILELIST` class

LISTBOX methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the list box and, optionally, its variable array component.

If `listbox.flags` does not contain `PR_LISTBOX_KEEP_ARRAY`, `listbox.va` (if not `NULL`) is sent a `DESTROY` message. The method then supersends the `DESTROY` message.

WN_INIT — Initialise

```
VOID wn_init(IN_LISTBOX *init);
```

Initialise the list box.

The horizontal cursor offset, `listbox.curoff`, is set to a suitable initial value of `LISTBOX_LEFT_EDGE + LISTBOX_OBLOID_INDENT`.

The list box data array handle, `listbox.va`, is set from `init->array`. This handle is assumed to be of an instance of a subclass of `VAROOT` and must always be supplied.

The value of `init->flags` is copied to `listbox.flags`. The significant flags for general behaviour of the list box are:

`IN_LISTBOX_KEEP_ARRAY`	Referred to in `listbox.flags` as `PR_LISTBOX_KEEP_ARRAY`. If set, prevents the `destroy` method from sending a `DESTROY` message to `listbox.va`.
`IN_LISTBOX_FORCE_WIDE`	Referred to in `listbox.flags` as `PR_LISTBOX_FORCE_WIDE`. If set, forces the list box width to include space for the scroll symbols that would otherwise only be shown if the list box contained more enties than could be displayed on the screen.
`IN_LISTBOX_WRAP_ROUND`	Referred to in `listbox.flags` as `PR_LISTBOX_WRAP_ROUND`. If set, pressing the up and down cursor keys will cause wrapping between the first and last entries in the list box.
`PR_LISTBOX_PLAQUE`	If set, the list box is displayed with a 3D border (not available on the Series 3). This is used, for example, for Help lists.

Further initialisation processing depends on other flags specified in `init->flags`. If set, these have the following meanings, explained in the order in which they are processed:

`IN_LISTBOX_MATCHER`	Create an instance of the `VMATCHER` class, storing its handle in `listbox.match`, and initialise it, effectively as follows: `p_send5(listbox.match,O_IM_INIT,&listbox.matchlen,128, init->array);`
`IN_LISTBOX_TEXT_OFFSET`	Set `listbox.offset` from `init->offset`, otherwise `listbox.offset` is zero
`IN_LISTBOX_FIRST_SET`	Set `listbox.first` from `init->first`, otherwise `listbox.first` is zero
`IN_LISTBOX_LAST_SET`	Set `listbox.last` from `init->last`, otherwise `listbox.last` is zero
`IN_LISTBOX_CUR_SET`	Set `listbox.current` from `init->current`, otherwise `listbox.current` is set equal to `listbox.first`
`IN_LISTBOX_MIN_WIDTH` or `IN_LISTBOX_FIXED_WIDTH`	Set the initial value of `listbox.width` from `init->minwid`, otherwise `listbox.width` is zero. The width of the list box will not be less than any initial value of `listbox.width`. If the flag `IN_LISTBOX_FIXED_WIDTH` is set (not on the Series 3) the width will not be further adjusted.

IN_LISTBOX_AUTO_SIZE	Send an LB_SIZE_WINDOW message, passing init->flags and the address of init->pos. This sets the height and (provided the width of the list box is not specified to be fixed) width of the list box to be sufficient to display the contents.

Note that if the IN_LIST_BOX_AUTO_SIZE flag is not set, the creator of an instance of LISTBOX must send an explicit LB_SIZE_WINDOW message before attempting to make the list box visible.

The remaining initialisation flags, IN_LISTBOX_POS_ALIGN_X and IN_LISTBOX_POS_ALIGN_Y, are of significance to the lb_size_window method.

WN_KEY — Handle key input

```
INT wn_key(INT keycode, INT modifiers);
```

Sends an LB_INQUIRE_LAST message to find the index number of the last item in the list box. Further processing depends on the value of keycode as follows:

W_KEY_RETURN	Returns listbox.current+1.
W_KEY_ESCAPE	Returns WN_KEY_CANCELLED.
W_KEY_LEFT, W_KEY_UP	If modifiers contains W_CTRL_MODIFIER, positions to the top item in the main list (with index number listbox.first). Otherwise, moves up one item in the list unless the current item is already the first. If already on the first item and listbox.flags contains PR_LISTBOX_WRAP_ROUND, positions to the last item in the list. Sets listbox.matchlen to zero, sets the new item by sending an LB_TAKE_FOCUS message and returns WN_KEY_NO_CHANGE.
W_KEY_RIGHT, W_KEY_DOWN	If modifiers contains W_CTRL_MODIFIER, positions to the last item in the main list. Otherwise, moves down one item in the list unless the current item is already the last. If already on the last item and listbox.flags contains PR_LISTBOX_WRAP_ROUND, positions to the first item in the list. Sets listbox.matchlen to zero, sets the new item by sending an LB_TAKE_FOCUS message and returns WN_KEY_NO_CHANGE.
W_KEY_PAGE_UP	If modifiers contains W_CTRL_MODIFIER, or if all the items are visible in the list box, positions to the first item in the main list. Sets listbox.matchlen to zero, sets the new item by sending an LB_TAKE_FOCUS message and returns WN_KEY_NO_CHANGE. Otherwise, scrolls up by one less than the number of visible items in the main list (or to the top of the list, if less) writing a new value to listbox.current, moving the emphasis to this item and, if listbox.match is not NULL, sending it an IM_SET_VAL message and resetting the match cursor to the first character of the item. The method then terminates by returning WN_KEY_NO_CHANGE.
W_KEY_PAGE_DOWN	If modifiers contains W_CTRL_MODIFIER, or if all the items are visible in the list box, positions to the last item in the main list. Sets listbox.matchlen to zero, sets the new item by sending an LB_TAKE_FOCUS message and returns WN_KEY_NO_CHANGE. Otherwise, scrolls down by one less than the number of visible items in the main list (or to the end of the list, if less) writing a new value to listbox.current, moving the emphasis to this item and, if listbox.match is not NULL, sending it an IM_SET_VAL message and resetting the match cursor to the first character of the item. The method then terminates by returning WN_KEY_NO_CHANGE.

`any other key` If `listbox.match` is `NULL`, keycode represents a printable character and `listbox.va` is not `NULL`, a cyclic search is made, forward from the current item, for a case-insensitive match between keycode and the first letter of an item. If a match is found, the matching item is made the current item by means of an `LB_TAKE_FOCUS` message.

If `listbox.match` is not `NULL`, incremental matching is attempted by sending `listbox.match` an `IM_KEY` message (which will adjust `listbox.matchlen` as appropriate) passing keycode and modifiers.

- if this returns `IM_NEW_DISPLAY`, the new current item is found by adding `listbox.vastart` to the return value from an `IM_SENSE_VAL` message sent to `listbox.match`. This item is made the current item by means of an `LB_TAKE_FOCUS` message
- if it returns `IM_NEW_LEN` the incremental matching cursor is redrawn in its new position, according to the value of `listbox.matchlen`

In all of these cases the method returns `WN_KEY_NO_CHANGE`.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

If `listbox.flags` contains `PR_LISTBOX_UNSTABLE` the method simply clears this flag and calls `wInvalidateWin` so that the list box will receive a `WN_REDRAW` message at a later time.

Otherwise the method draws the list box border and sends an `LB_INQUIRE_LAST` message to find the index number of the last item to be drawn. It then draws its content as follows:

- if `listbox.first` is not zero the items with index numbers from 0 to `listbox.first-1` are drawn at the top of the list box, followed by a horizontal line across the full width of the list box.
- if `listbox.top` is greater than `listbox.first`, indicating that more items are available above the first visible item in the main list, an up arrow is drawn at the top of the scroll gutter
- starting with the item with index number `listbox.top`, successive items are drawn until the items are exhausted or the list box is full. If the item that has just been drawn is the selected item (with index number equal to `listbox.current`) and `win.flags` contains `PR_WIN_EMPHASISED`, the item is emphasised by sending an `LB_DRAW_EMPHASIS` message
- if more items are available following the last item drawn, a down arrow is drawn at the bottom of the scroll gutter

The text for each item is found by sending an `LB_INQUIRE_ITEM` message and is drawn by sending an `LB_DRAW_ITEM` message.

WN_EMPHASISE — Set emphasis

```
VOID wn_emphasise(INT flag);
```

Set the list box window's emphasis if `flag` is `TRUE`, otherwise clear the emphasis.

The method first sets or clears the `PR_WIN_EMPHASISED` flag in `win.flags`, creates a temporary graphics context and draws the border in the appropriate emphasised state.

Provided `listbox.current` is greater than or equal to `listbox.first`, the method calculates, in a `P_RECT` struct, the rectangular region within its window corresponding to the `listbox.current` item. It then sends an `LB_DRAW_EMPHASIS` message, passing `listbox.current`, a pointer to the `P_RECT` struct and the value of `flag`.

The temporary graphics context is then freed. If incremental matching is being used the incremental text cursor is drawn in its current position if `flag` is `TRUE`, otherwise it is erased.

LB_SIZE_WINDOW Set the window size and position

```
VOID lb_size_window(INT flags, P_POINT *pos);
```

Set the size of the window to that required to display the contents. The position of the window's top left corner is set, guided by the flags passed in `flags` and the coordinates in the `P_POINT` struct pointed to by `pos`.

Note that this method includes the code that creates the window and so *must* be called before attempting to make the list box visible. It will normally be called from the `wn_init` method, provided the parameters to this method include the `IN_LISTBOX_AUTO_SIZE` flag.

On entry, `listbox.width` contains a previously specified minimum allowable width for the list box. Provided `listbox.flags` does not include `PR_LISTBOX_FIXED_WIDTH`, the required width to display all the list box items is calculated as the larger of:

- the value of `listbox.width` less the left and right borders
- the widest of all the items, found by sending an `LB_ITEM_WIDTH` message for each item

This value overwrites the original value of `listbox.width`, but the original value is also held for later use. The total width of the window is calculated by adding `listbox.width`, `LISTBOX_LEFT_EDGE`, `LISTBOX_LEFT_EDGE` and (if either `listbox.flags` contains `PR_LISTBOX_FORCE_WIDE`, or there are more than seven items to be displayed) `LISTBOX_SCROLL_GUTTER`. Except in the case of the Work*about*, if the calculated width exceeds the screen width, the method calls `p_leave(E_GEN_TOOWIDE)`.

The height of the list box is calculated to display a the number of items in the list, provided this does not exceed the number of items that can be shown on the screen. If `listbox.first` is not zero, the height is increased by the additional amount required for the line dividing the title from the main list.

Once the dimensions of the list box have been determined, the position on the screen is calculated. Three options are allowed:

- if `flags` contains `IN_LISTBOX_POS_ALIGN_X` the list box is centred horizontally on the region starting as `pos->x` and of width equal to the value that `listbox.width` had on first entry to the method. If this results in the window extending beyond either the left or right edge of the screen, the window is positioned at the corresponding edge. The y-coordinate of the top left corner of the list box window is set to be equal to `pos->y`. This option is used, for example, to display a pull-down menu, centred as closely as possible on its menu bar item.
- if `flags` contains `IN_LISTBOX_POS_ALIGN_Y` the list box is positioned so that, as far as possible, the currently highlighted item (the item with index `listbox.current`) is aligned with the position indicated by `pos->y`. The x-coordinate of the top left corner of the list box window is set so that the text of the items starts at the position indicated by `pos->x`, unless this results in the window exrending beyond the right edge of the window, in which case it is positioned as far to the right as possible. This option is used, for example, to display the list used to display an expanded set of choices for a dialog box choice list control.
- if `flags` contains neither `IN_LISTBOX_POS_ALIGN_X` nor `IN_LISTBOX_POS_ALIGN_X`, the list box is centred on the screen by sending a `WN_CALC_POSITION` message.

The list box's `win.flags` is set to `PR_BWIN_CUSHION|PR_WIN_EMPHASISED` ORed with either `IN_BWIN_SHADOW_1` or, if `listbox.flags` contains `PR_LISTBOX_PLAQUE`, `IN_BWIN_SHADOW_2`. The window is created with the required position and size as a root window (that is, with a `NULL` value of the parameter `par`) by sending a `WN_CONNECT` message.

If incremental matching is enabled, the incremental matching cursor is drawn at the first character of the currently selected item.

LB_DRAW_ITEM Draw an item

```
VOID lb_draw_item(TEXT *buf, INT index, P_RECT *prect);
```

Draw, in the rectangle specified by `prect`, the content for the list box item with index number `index`, using the zero terminated text pointed to by `buf`.

The text is drawn to the current graphics context by a call to the window server function `gPrintBoxText`:

It is the caller's responsibility to ensure that a suitable graphics context exists.

This method does not use index, which is provided for subclasses that replace this method (PULLDOWN, HELPLIST and FILELIST).

LB_DRAW_EMPHASIS — Toggle emphasis for an item

```
VOID lb_draw_emphasis(INT index, P_RECT *prect, INT on);
```

Toggle the presence of a highlighting obloid, in the rectangle specified by prect, for the item with index number index. The value of on is TRUE if emphasis is being set, and FALSE if emphasis is being cleared.

The action is as indicated in the following code:

```
METHOD VOID listbox_lb_draw_emphasis(PR_LISTBOX *self,INT index,P_RECT *prect,INT on)
    {
    P_EXTENT ob;

    ob.tl=prect.tl;
    ob.width=p_send3(self,O_LB_ITEM_WIDTH,index);
    ob.height=DatGate->gate.x.cht;
    gInvObloid(&ob);
    }
```

The supplied method does not use on, which is provided for use by the HELPLIST subclass.

LB_ITEM_WIDTH — Sense required width for an item

```
INT lb_item_width(INT index);
```

Return the pixel width required to display the text of the item with index number index, including a gap of width LISTBOX_OBLOID_INDENT at either side of the text, to allow the item to be highlighted.

LB_TAKE_FOCUS — Move focus to an item

```
VOID lb_take_focus(INT focus);
```

Select the item with index number focus.

If focus is the index number of the item that is currently selected the method does nothing other than ensure that, provided listbox.match is not NULL and the window is emphasised (win.flags contains PR_WIN_EMPHASISED) the incremental matching cursor is drawn in its current position.

Otherwise, listbox.current is set to focus. Provided the window is emphasised, the previously selected item has its highlighting removed and the highlight is set on the item with index number focus. If necessary, the list box contents are scrolled until this item is visible.

If incremental matching is enabled, listbox.match is sent an IM_SET_VAL message, passing the value of focus-listbox.vastart and, provided the window is emphasised, the incremental matching cursor is drawn in its newly set position.

LB_INQUIRE_FOCUS — Sense the item with focus

```
INT lb_inquire_focus(VOID);
```

Return the index number of the currently selected item.

This method simply returns the value of listbox.current.

LB_INQUIRE_ITEM — Get pointer to item text

```
TEXT *lb_inquire_item(INT index);
```

Return a pointer to the first character of the text of the item with index number index.

The pointer is calculated by adding listbox.offset to the pointer returned by sending the message:

```
p_send3(self->listbox.va,O_VA_PBUF,index);
```

There is an implicit assumption that the entire content of the item is stored contiguously.

LB_INQUIRE_LAST — Get index of last item

```
INT lb_inquire_last(VOID);
```

Return the index number of the last item in the list box.

If `listbox.last` is not zero, return this value. Otherwise return one less than the number returned by a VA_COUNT message sent to `listbox.va`.

The Menu Bar

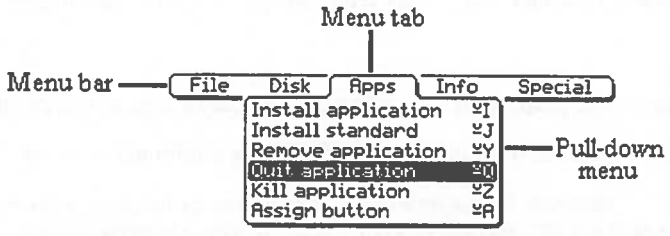
WIN	BWIN
flags id	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ wn_emphasise

MENUBAR
tab menu mb num xoff toff pos
destroy wn_init wn_key wn_draw wn_visible mb_add_menu

An HWIM menu bar provides a means of browsing through an application's commands and their associated command accelerators. In addition, it may be used to select and execute a command, as an alternative to the direct use of an accelerator.

A menu bar uses component instances of the MENUTAB and PULLDOWN classes. Note that one or other of a menu bar's pull-down menus is always displayed while a menu bar is visible, and one item, the current item, is highlighted. The command corresponding to the current item may be executed by pressing Enter.

A typical menu bar (from the Series 3 System Screen) is shown below, together with a MENUTAB and a PULLDOWN component.



In an HWIM application the menu bar and its components do not exist unless the menu bar is visible. They are created each time the menu bar becomes visible (generally from the WSERV ao_run method in response to the user pressing the Menu key) and destroyed when it disappears (when the user either presses Esc or executes a command).

When the menu bar is displayed, one menu is pulled down (the *Apps* menu in the above example) and an item in that menu (for example, *Quit application*) is selected. The initial visible menu and selected item are determined by the WSERV property items accessed by `w_ws->wserv.info->menu` and `w_ws->wserv.info->mnitem` respectively.

An HWIM application will normally rely on system code to manage the presentation of the menu bar. Applications are not expected to subclass MENUBAR or to send explicit messages to any instance. The following description is included for completeness, and for interest.

Class definition

Defined in sub-category file *menubar.cl* (generated header file *menubar.g*).

```
CLASS menubar    bwin
The menu bar
  {
  REPLACE destroy          Free any memory used
  REPLACE wn_init          Set up the menu bar text and menus to pull down
  REPLACE wn_emphasise     For 3d border effects
  REPLACE wn_draw          Draw the menu bar text
  REPLACE wn_key           Toggle through the menus or pass on to menus
  REPLACE wn_visible       When made visible show menu as well
  ADD mb_add_menu          Add a menu to the menu bar

  TYPES
    {
    typedef struct
      {
      UWORD menu_id;  resource ID of associated pull-down menu
      TEXT mb_txt[1]; menu header text (ZTS)
      } MENUBAR_ITEM;
    typedef struct
      {
      UBYTE count;    number of items
      } MENUBAR;      (followed by count MENUBAR_ITEMs)
  }

  CONSTANTS
    {
    MENUBAR_LEFT_SHOULD   4  horizontal space occupied by the menu bar left shoulder
    MENUBAR_RIGHT_SHOULD  4  horizontal space occupied by the menu bar right shoulder
    MENUBAR_MAX_MENU      15
    MENUBAR_MIN_GAP       3  half the minimum separation between successive text items
    MENUBAR_MAX_GAP       10 half the maximum separation between successive text items
    MENUBAR_3b_TAIL_WIDTH 18
    }

  PROPERTY 2
    {
    PR_MENUTAB *tab;     Tab top for current pull down menu
    PR_PULLDOWN *menu;   Current pull down menu
    MENUBAR *mb;         Pointer to menu bar information
    UBYTE num;           Number of currently pulled down menu
    UBYTE xoff;          X offset of start of menu bar on screen
    UBYTE toff;          Offset of start of text from tab position
    UBYTE asc;           Ascent where text should be drawn
    UWORD pos[MENUBAR_MAX_MENU+1];  Leading tab positions, including terminal entry
    }
  }
```

On the Series 3, the `menubar.asc` property is not defined and `menubar.pos` is a UBYTE array.

The Series 3 version of the class definition includes the following additional CONST definitions:

```
MENUBAR_MAX_WIDTH     (SCREEN_WIDTH-MENUBAR_LEFT_SHOULD-MENUBAR_RIGHT_SHOULD)
MENUBAR_HEIGHT   (SYSTEM_FONT_HEIGHT+(BWIN_CUSHION_Y+2)*2+BWIN_SHADOW_1_HEIGHT)
```

On the Series 3a the corresponding values are read or derived from the globally accessible data that is described in the *Introduction* chapter of this manual.

Property	
menubar.tab	The handle of the menu tab (an instance of MENUTAB)for the currently displayed pull-down menu.
menubar.menu	The handle of the currently displayed pull-down menu.
menubar.mb	A pointer to a MENUBAR struct, containing an array of MENUBAR_ITEM structs, one for each of the menu bar's pull-down menus.
menubar.num	The index number of the currently visible pull-down menu.
menubar.xoff	The screen x-offset, in pixels, to the left hand edge of the menu bar. This is stored for convenience, rather than repeatedly reading it from the window server.
menubar.toff	The pixel offset from the left edge of any menu tab to the start of the corresponding text.
menubar.asc	The ascent, in pixels, to the position where the menubar text is drawn. This differs for different machine types. This property is not defined on Series 3 machines.
menubar.pos	An array of the positions of the menu tabs for the tops of all the pull-down menus. Each entry contains the pixel position, relative to the left edge of the menu bar, of the left edge of the menu tab of the corresponding pull-down menu.The array contains a terminating entry that indicates the rightmost extent of the last menu tab. On the Series 3, menubar.pos is an array of UBYTES.

MENUBAR methods

Displaying a pull-down menu

A pull-down menu is displayed by creating and initialising component instances of the MENUTAB and of PULLDOWN or (if a 3D border is used on the Series 3a) XPULLDWN classes. An initial pull-down menu is set up by the wn_visible method, and the wn_key method may subsequently switch to another menu.

The pull-down menu to be displayed is identified by the value of menubar.num, which is used to locate the corresponding MENUBAR_ITEM struct in the buffer pointed to by menubar.mb.

If there are existing MENUTAB and PULLDOWN components they are destroyed before new instances are created, with their handles stored in menubar.tab and menubar.menu.

The PULLDOWN instance is initialised from an IN_LISTBOX list box initialisation struct. This struct is defined in the LISTBOX class definition as:

```
typedef struct
      {
      UWORD flags;
      PR_VAROOT *array;
      UWORD offset;
      WORD current;
      UWORD top;
      UWORD first;
      UWORD last;
      P_POINT pos;
      UWORD minwid;
      } IN_LISTBOX;
```

where:

- flags is set to
  IN_LISTBOX_TEXT_OFFSET | IN_LISTBOX_MIN_WIDTH | IN_LISTBOX_WRAP_ROUND |
  IN_LISTBOX_POS_ALIGN_X | IN_LISTBOX_AUTO_SIZE | IN_LISTBOX_CUR_SET
- array contains the handle of an instance of VARES, containing the items loaded from the pull-down menu resource specified by the value of menu_id in the relevant MENUBAR_ITEM struct

- offset is set to 1 to skip the first byte (containing the command id) of each element of the array
- current is set to either the value of w_ws->wserv.info->mnitem (on creation of the menu bar) or zero (when switching to another menu)
- top, first and last are not used
- pos.x is set to the value of menubar.pos[menubar.num] less the value MENUBAR_LEFT_SHOULD and pos.y is set to be four pixels less than the height of the menubar
- minwid is set to the width of the menu tab plus an allowance for the left and right shoulders of the pull-down menu

The command manager is sent a COM_MENU message (to inform it of the imminent display of the pull-down menu) passing menubar.num and the array handle. The pull-down menu is then sent a WN_INIT message, passing the address of the IN_LISTBOX struct described above. The menu tab is also sent a WN_INIT message, passing the address of the P_EXTENT struct described earlier, the value of menubar.toff and a pointer to the menu bar text for that item.

Finally both the pull-down menu and the menu tab are sent a WN_VISIBLE,WV_INITVIS message.

The code to implement the display of a pull-down menu is executed under the protection of p_enter and returns either zero on successful completion or a negative error number on failure.

DESTROY — Destroy

```
VOID destroy(VOID);
```

Record the current pull-down menu item selection by writing menubar.num to w_ws->wserv.info->menu and menubar.menu->listbox.current to w_ws->wserv.info->mnitem. If w_ws->wserv.oldinfo is not NULL (normally meaning that the current menu is a submenu) w_ws is sent a WS_RESET_MENUBAR message, passing w_ws->wserv.oldinfo, and then w_ws->wserv.oldinfo is set to NULL.

The menu bar is destroyed by freeing the allocated cell pointed to by menbar.mb and then supersending the DESTROY message. Finally, w_ws->wserv.bar is set to NULL.

WN_INIT — Initialise

```
INT wn_init(MENUBAR *pmenbar);
```

Initialise the menu bar to display the menu items in the MENUBAR struct pointed to by pmenbar (this struct will normally have been loaded from a resource file).

The method scans the text of the items in the MENUBAR struct, calculating the cumulative pixel position of each item and storing the results in successive elements of the menbar.pos array. The text of successive items is separated by 2*MENUBAR_MIN_GAP pixels. Except in the case of the Work*about*, if the total width of the menu bar exceeds the width of the screen (less the sum of MENUBAR_LEFT_SHOULD, MENUBAR_RIGHT_SHOULD and TAB_EDGES) the method calls p_leave(E_GEN_TOOWIDE).

The value of win.flags is set to PR_BWIN_SHADOW|PR_BWIN_CUSHION|PR_WIN_EMPHASISED and the window is created as a top-level window by sending a WN_CONNECT message. The method does not set the position or dimensions of the window. This must be done by the later sending of a WN_VISIBLE message.

The method returns zero to indicate that p_leave has not been called, and is thus suitable for being called under the protection of p_enter.

WN_VISIBLE — Set visibility

```
INT wn_visible(UINT state);
```

Set the position and size of the menu bar and make it (and one of its pull-down menus) visible. The implementation of the method assumes that a menu bar will receive only one WN_VISIBLE message in its lifetime, and that the value of state will be WV_INITVIS.

To recover any previous selection of a pull-down menu, menubar.num is set to w_ws->wserv.info->menu. This pull-down menu will be drawn with a highlight set to the item indicated by w_ws->wserv.info->mnitem. Both of these values are protected against overrun, should the number of

pull-down menus or the number of items within a pull-down menu be decreased since the last time the menu bar was visible.

The gaps between the menu bar items are expanded as much as possible, up to a maximum of `2*MENUBAR_MAX_GAP`. This determines the width of the menu bar, which is then centred horizontally and the pixel position of the left edge of the menu bar is stored in `menubar.xoff`. The menu bar is positioned at the top edge of the screen, with a fixed height equal to the height of the menu bar. The position and dimensions of the menu bar are set by a call to `wSetWindow`. The value of `menubar.toff` is set so that the text of each menu bar item will appear centrally within its menu tab.

The method supersends the `WN_VISIBLE` message before displaying the pull-down menu with index number `menubar.num`, highlighting the item with index number `w_ws->wserv.info->mnitem`.

The method returns either zero on successful completion or a negative error number. Errors will be related to a failure to create and display the pull-down menu.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the menu bar border and the text of each item.

Each item of text is drawn by means of a call to `gPrintText`. The horizontal position of each text item is determined by adding `menubar.toff` to the value of corresponding entry in the `menubar.pos` array.

WN_KEY — Handle key input

```
INT wn_key(UINT keycode,UINT modifiers);
```

Process the keypress with key code `keycode` and the modifiers indicated by `modifiers`. The following table describes the response to specific keys.

`W_KEY_LEFT`	Cycle left by one item and display the new pull-down menu. The return value is either `WN_KEY_NO_CHANGE` (0) or a negative error caused by a failure to create and display the pull-down menu.
`W_KEY_RIGHT`	Cycle right by one item and display the new pull-down menu. The return value is either `WN_KEY_NO_CHANGE` (0) or a negative error caused by a failure to create and display the pull-down menu.
`W_KEY_MENU`	If modifiers contains `W_SHIFT_MODIFIER` perform the same action as for `W_KEY_LEFT`, otherwise perform the same action as for `W_KEY_RIGHT`.
`W_KEY_HOME`	Display the first pull-down menu. The return value is either `WN_KEY_NO_CHANGE` (0) or a negative error caused by a failure to create and display the pull-down menu.
`W_KEY_END`	Display the last pull-down menu. The return value is either `WN_KEY_NO_CHANGE` (0) or a negative error caused by a failure to create and display the pull-down menu.
`numeric, 1 to 9`	Display the corresponding pull-down menu; numbers greater than the number of items in the menu bar all display the last pull-down menu. The return value is either `WN_KEY_NO_CHANGE` (0) or a negative error caused by a failure to create and display the pull-down menu.

All other key codes for which `p_isprint` returns `FALSE` are sent, via a `WN_KEY` message, to the currently displayed pull-down menu, whose handle is in `menubar.menu`. Significant key codes are `W_KEY_ESCAPE`, `W_KEY_RETURN`, `W_KEY_UP`, `W_KEY_DOWN`, `W_KEY_PAGE_UP` and `W_KEY_PAGE_DOWN`. The return value is the value returned by the `WN_KEY` message to the pull-down menu.

All remaining key codes are tested for a case-independent match with the accelerator keys, stored in `w_ws->wserv.info.accel`. If there is a match, and the corresponding command is in the currently displayed pull-down menu, the pull-down menu is sent an `LB_TAKE_FOCUS` message to highlight the appropriate item. If the match is with a command in another pull-down menu, the new pull-down menu is displayed, with the appropriate item highlighted. The return value is either `WN_KEY_NO_CHANGE` (0) or a negative error caused by a failure to create and display a pull-down menu.

MB_ADD_MENU — Add a menu

```
INT mb_add_menu(TEXT *title,INT menu_id);
```

Append a pull-down menu to the end of the menu bar. The additional item will be displayed with the text in the zero terminated string pointed to by `title` and uses the pull-down menu indicated by the resource ID `menu_id`. If used, this method may only be called between the calls to the `wn_init` and `wn_visible` methods.

Uses `p_realloc` to reallocate the buffer pointed to by `menubar.mb` so that it is large enough to contain the additional item. The text string and `menu_id` are copied into `menubar.mb` to form its final `MENUBAR_ITEM` struct, an additional entry is added to the `menubar.pos` array (as with the `wn_init` method, the inter-item gap is set to `2*MENUBAR_MIN_GAP`) and `menubar.mb->count` is incremented.

The method returns zero if it completed successfully, or one of the errors:

`E_GEN_NOMEMORY`	if there was insufficient memory to reallocate the buffer
`E_GEN_TOOWIDE`	if the new width is too wide for the screen (this error is not returned on the Work*about*)
`E_GEN_TOOMANY`	if the number of menus exceeds `MENUBAR_MAX_MENU` (this error is not returned on the Work*about*)

PULLDOWN pull-down menu

WIN	BWIN	LISTBOX	PULLDOWN
flags id		match va flags width matchlen curoff offset current top first last vastart matchstart	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help wn_visible *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ ~~wn_emphasise~~	destroy wn_init ~~wn_key~~ wn_draw wn_emphasise ~~lb_draw_item~~ ~~lb_draw_emphasis~~ lb_size_window ~~lb_item_width~~ lb_take_focus lb_inquire_focus lb_inquire_item lb_inquire_last	wn_key lb_draw_item lb_draw_emphasis lb_item_width

The `PULLDOWN` class subclasses `LISTBOX` to provide the display of pull-down menus. In an HWIM application a pull-down menu does not exist unless it is visible.

An HWIM application will normally rely on system code to manage the presentation of pull-down menus. Applications are not normally expected to subclass PULLDOWN or to send explicit messages to any instance. The following description is included for completeness, and for interest.

Class definition

Defined in sub-category file *pulldown.cl* (generated header file *pulldown.g*).

```
CLASS   pulldown        listbox
pull down menu from menu bar
    {
    REPLACE wn_key              Check for accelerator matching
    REPLACE lb_draw_item        Draw accelerator at right edge
    REPLACE lb_draw_emphasis    Emphasise whole area
    REPLACE lb_item_width       Returns the required width of the item

    TYPES
        {
        typedef struct
            {
            UBYTE com_id;
            TEXT mn_txt[1];
            } MENU_ITEM;
        }
    }
```

Property

There is no property associated with the PULLDOWN class.

PULLDOWN methods

WN_KEY — Handle key input

```
INT wn_key(INT keycode,INT modifiers);
```

Returns the value returned by supersending the WN_KEY message, provided the value is zero (WN_KEY_NO_CHANGE) or negative (WN_KEY_CANCELLED).

The behaviour is thus similar to that of the LISTBOX superclass wn_key method, except for its response to the W_KEY_ENTER value of keycode (for which the superclass method returns the index number of the current item, plus one).

In this case the PULLDOWN wn_key method returns the value of the first byte of the array entry for the currently selected menu item, obtained by sending listbox.va a VA_PBUF message. This byte contains the command ID of the command manager method corresponding to the menu item (any Series 3a BREAK_LINE_FOLLOWS flag is removed).

LB_DRAW_ITEM — Draw an item

```
VOID lb_draw_item(TEXT *txt,INT index,P_RECT *area);
```

Draw the item with index number index within the area specified by area and containing the text pointed to by txt. This text is assumed to be part of a MENU_ITEM struct, that is, the byte immediately preceding that pointed to by txt is assumed to contain the ID of the corresponding menu command.

The menu item text is drawn by supersending the LB_DRAW_ITEM message. Following this the accelerator text is constructed from the WS_SYMBOL_PSION character and the (uppercased) result of a WS_SENSE_ACCEL message sent to the application's instance of WSERV. This text is then printed in the item by means of a call to gPrintText.

For Series 3a machines, if the accelerator is an upper case character the shift key indicator (in English, the text "Shift") is displayed before the accelerator text. If the command ID contains the Series 3a flag BREAK_LINE_FOLLOWS, a grey line is drawn below it.

LB_DRAW_EMPHASIS — Set item emphasis

```
VOID lb_draw_emphasis(INT index,P_RECT *prect,INT on);
```

Toggle the presence of a highlighting obloid, in the rectangle specified by `prect`, for the item with index number `index`. The value of on is `TRUE` if emphasis is being set, and `FALSE` if emphasis is being cleared.

The action is similar to that of the `lb_draw_emphasis` method of the `LISTBOX` superclass, except that the width of the highlighting obloid is not found by sending an `LB_ITEM_WIDTH` message, but is set to the full width of the rectangle specified by `prect`. It thus includes the accelerator key text.

As for the `LISTBOX` `lb_draw_emphasis` method, the supplied method does not use `on`.

LB_ITEM_WIDTH — Sense required width for an item

```
INT lb_item_width(INT index);
```

Returns the width required to display the item with index number `index`.

The required width found by adding the space required to display the accelerator text to the value returned by supersending the `LB_ITEM_WIDTH` message.

XPULLDWN pull-down menu

WIN	BWIN	LISTBOX	PULLDOWN	XPULLDWN
flags id		match va flags width matchlen curoff offset current top first last vastart matchstart		
~~destroy~~ wn_calc_position ~~wn_connect~~ wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help wn_visible *wn_set* *wn_sense* ~~wn_draw~~ ~~wn_init~~	~~wn_draw~~ ~~wn_emphasise~~	destroy wn_init ~~wn_key~~ ~~wn_draw~~ wn_emphasise ~~lb_draw_item~~ ~~lb_draw_emphasis~~ lb_size_window ~~lb_item_width~~ lb_take_focus lb_inquire_focus lb_inquire_item lb_inquire_last	wn_key lb_draw_item ~~lb_draw_emphasis~~ lb_item_width	wn_connect wn_draw lb_draw_emphasis

The `XPULLDWN` class subclasses `PULLDOWN` to provide the Series 3a version of the display of pull-down menus. This class is not present on Series 3 machines.

An HWIM application will normally rely on system code to manage the presentation of pull-down menus. Applications are not normally expected to subclass `XPULLDWN` or to send explicit messages to any instance. The following description is included for completeness, and for interest.

Class definition

Defined in sub-category file *xpulldwn.cl* (generated header file *xpulldwn.g*).

```
CLASS xpulldwn pulldown
    {
    REPLACE wn_connect
    REPLACE wn_draw
    REPLACE lb_draw_emphasis
    CONSTANTS
        {
        PULLDOWN_3b_LEFT        4    additional space required by S3a menus
        PULLDOWN_3b_RIGHT       3
        PULLDOWN_3b_BOTTOM      3
        PULLDOWN_3b_TOP         4
        }
    }
```

Property

There is no property associated with the XPULLDWN class.

XPULLDWN methods

WN_CONNECT — Connect to the window server

```
VOID wn_connect(PR_WIN *par, UINT flags, W_WINDATA *pwd);
```

The window dimensions, in pwd->ext, are adjusted to the slightly larger dimensions of the Series 3a pull-down menus, and the result adjusted if necessary to ensure that the window does not exceed the boundaries of the screen. The flags W_WIN_BACK_CLR and W_WIN_BACK_GREY_CLR are ORed into pwd->background, and W_WIN_BACKGROUND is ORed into flags. The method then supersends the WN_CONNECT message.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the pull-down menu border and the text of each item.

Each item of text is drawn by means of a call to gPrintText. The currently selected item is highlighted by dending an LB_DRAW_EMPHASIS message.

LB_DRAW_EMPHASIS — Set item emphasis

```
VOID lb_draw_emphasis(INT index,P_RECT *prect,INT on);
```

Toggle the presence of a highlighting obloid, in the rectangle specified by prect, for the item with index number index. The value of on is TRUE if emphasis is being set, and FALSE if emphasis is being cleared.

Increment the x-coordinates in the P_RECT struct pointed to by prect and supersend the LB_DRAW_EMPHASIS message.

As for the superclass method, the value of on is ignored.

MENUTAB

WIN		BWIN
flags id		
destroy wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ wn_key wn_position wn_redraw	wn_sense_help wn_visible *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ wn_emphasise

MENUTAB
txt xoff
wn_init wn_draw

The MENUTAB class subclasses BWIN to provide the display of the tab above a currenly visible pull-down menu. In an HWIM application the tab does not exist unless it, and its associated pull-down menu, is visible.

An HWIM application will normally rely on system code to manage the presentation of a menu tab. Applications are not normally expected to subclass MENUTAB or to send explicit messages to any instance. The following description is included for completeness, and for interest.

Class definition

Defined in sub-category file *menubar.cl* (generated header file *menubar.g*).

```
CLASS menutab    bwin
    {
    REPLACE wn_init
    REPLACE wn_draw
    PROPERTY
        {
        TEXT *txt;          Pointer to the text to display
        UWORD xoff;         Offset at which to display the text
        }
    }
```

Property

`menutab.txt`	A pointer to the menu tab text, assumed to be a zero terminated string.
`menutab.xoff`	The pixel x-offset in the menu tab window to the position at which the text is to be drawn.

MENUTAB methods

WN_INIT — Initialise

```
VOID wn_init(P_EXTENT *pext,UINT xoff,TEXT *txt);
```

Sets `win.flags` to `PR_WIN_EMPHASISED|PR_BWIN_CUSHION|PR_BWIN_SHADOW_1|PR_BWIN_OPEN`, `menutab.off` to `xoff` and `menutab.txt` to `txt`. Then sends a WN_CONNECT message to create the window with position and size as specified by `pext`.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the menu tab.

Supersends the WN_DRAW message to draw the border and then draws the text of the menu tab with a call to `gPrintText`.

CHAPTER 7

DIALOG BOXES

This chapter describes the classes provided in HWIM for the creation and operation of dialogs. A dialog box shows the user the current values of one or more data items and, in general, allows the user to modify one or more of these values.

In an HWIM application the most common use of a dialog is as a result of the user selecting a command from a command menu. In response to an *Open file* command, for example, a dialog would be presented to allow the user to specify the name (and possibly the type) of the file to be opened.

All HWIM dialogs are *modal*, that is, while the dialog is visible the user can interact with the application only via that dialog; the application enters a 'mode' such that all attempts to interact with, say the menu bar are disallowed. This mode terminates when the user satisfactorily completes the dialog.

A number of pre-defined system dialogs exist and are described in later chapters of this manual. The system dialogs may be run by specific WSERV methods, such as `ws_error_dialog`, `ws_query_dialog` and `ws_format_dialog`, that are described in the chapter *The WSERV Class*.

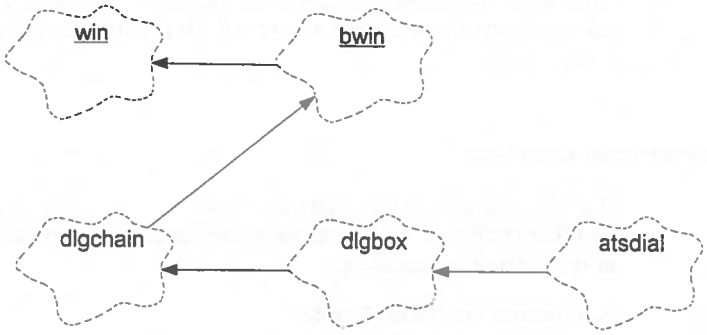
See also the dialog utility functions described in the *Dialog box utilities* section of the *HWIM Utility Functions* chapter.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the window concepts described in the *Introduction* and *Windows* chapters of the *Window Server Reference* manual
- the WSERV class, especially the `ws_do_dial` method, which starts up a dialog box, and the `ao_run` method, which may send messages to a dialog box window
- graphics contexts and drawing, described in the *Graphics Output* chapter of the *Window Server Reference* manual
- the basic principles of resource files as described in the *Resource Files* chapter of the *Additional System Information* manual. (Further information is available in the *HWIM Resource Files* chapter of the *Object Oriented Programming Guide*.)

Class diagram



DLGCHAIN

WIN		BWIN
flags	id	
destroy wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ wn_key wn_position wn_redraw	wn_sense_help wn_visible *wn_set* *wn_sense* ~~*wn_draw*~~ *wn_init*	wn_draw wn_emphasise

DLGCHAIN
next

The DLGCHAIN abstract class subclasses BWIN, but adds or modifies no methods. It adds property that makes all instances of its subclasses suitable for including in a chained list of objects. Such a list is maintained by the application's instance of WSERV (with the foremost dialog's handle stored in `wserv.dial`) to implement stacked dialog boxes.

Other subclasses of DLGCHAIN may also be included in this list. For example, Help 'dialogs' (which do *not* subclass DLGBOX) may appear in the list, intermixed with true dialogs.

Class definition

Defined in sub-category file *dlgbox.cl* (generated header file *dlgbox.g*).

```
CLASS   dlgchain        bwin
Dialog box next and previous chaining
    {
    CONSTANTS
        {       ! All four bits re-used from original meanings
        PR_WIN_NO_DDP  PR_WIN_FORCE_TOP DatDialogPtr should never point to this dialog
        HELPDLG_BASIC_HELP  PR_WIN_FORCE_RIGHT
        HELPDLG_HELP_INDEX  PR_WIN_FORCE_LEFT
        DLGCHAIN_WITH_MENU  PR_WIN_FORCE_BOTTOM
        }
    PROPERTY
        {
        PR_BWIN *next;          Next dialog, if present
        }
    }
```

On the Series 3 the CONSTANTS section of the class definition does not contain the definitions:

```
        HELPDLG_BASIC_HELP
        HELPDLG_HELP_INDEX
        DLGCHAIN_WITH_MENU
```

Property

`dlgchain.next` — The handle of the next item in a list of DLGCHAIN instances, mainly used for the list headed by the item whose handle is stored in the WSERV property `wserv.dial`.

DLGCHAIN flags

The following flags may be set in `win.flags`:

`PR_WIN_NO_DDP` — If a subclass of DLGCHAIN sets the `PR_WIN_NO_DDP` flag in `win.flags` during its initialisation, it will never have the handle of an instance written to the magic static `DatDialogPtr`.

`HELP_DLG_BASIC_HELP` — Not defined for Series 3 code.

`HELPDLG_HELP_INDEX` — Not defined for Series 3 code.

DLGCHAIN_WITH_MENU	Not defined for Series 3 code. If set, indicates that the dialog has an associated menu bar whose commands are accessible by the Menu and cursor keys, or by hot-key combinations. If, as is the case for most dialogs, the flag is clear, then all such keypresses are directed to the dialog box itself.

If a subclass does not set PR_WIN_NO_DDP in win.flags, it is signalling that it is prepared to have DatDialogPtr set to the handle of any instance that appears in the wserv.dial list. An advantage of doing so is that the handle does not need to be passed as a parameter to any utility functions - see, for example, the dialog box utility functions described later in this chapter. The subclass should write its handle to DatDialogPtr when it adds itself at the front of the wserv.dial list, during initialisation. Note that this handle may be overwritten by a different value when another subclass of DLGCHAIN adds itself to the list.

Any such subclass must cooperate with other items in the wserv.dial list. On destruction, provided DatDialogPtr contains its handle, it must scan all the remaining items in the list and write to DatDialogPtr the handle of the first item for which win.flags does not contain PR_WIN_NO_DDP.

Note that this means that any such instance is liable to have its handle written to DatDialogPtr, and thus must perform the operation described above, even if it does not write its own handle to DatDialogPtr. This situation will never arise for a subclass that sets PR_WIN_NO_DDP.

DLGBOX

WIN	BWIN	DLGCHAIN
flags id		next
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~	

DLGBOX	
item	count
rbuf	current
dimrid	font*
helprid	absorb
flags	changed
focus	
destroy	dl_item_replace
wn_key	dl_item_append
wn_emphasise	dl_init
wn_sense_help	dl_dimmed_message
wn_set	dl_item_add
wn_sense	dl_set_size
wn_draw	dl_inq_minsize
dl_item_lock	dl_dyn_init
dl_item_dim	dl_key
dl_set_item_flags	
dl_set_prompt	*dl_changed*
dl_take_focus	*dl_focus*
dl_handle_to_index	*dl_launch_sub*
dl_index_to_handle	*dl_item_new*

The DLGBOX class provides a flexible set of mechanisms for displaying and controlling a wide variety of dialog boxes. Although formally an abstract class (since some methods are DEFERred) an instance of DLGBOX may be created and used to display simple notification dialogs. The majority of normal dialogs may be created and used with a subclass that replaces at most two methods: very few dialogs require all the DEFERred methods to be defined. This is discussed at greater length in the *Using dialog boxes* section of the *Object Oriented Programming Guide*.

An HWIM dialog box consists of a bordered window containing between one or more lines, or items. The maximum bumber of items that may be displayed varies from machine to machine as follows:

Series 3	seven items
Series 3a	nine items
Work*about*	six items in the system font, or eight items in a small font

Each item may be plain text, a control, or a combination of a plain text prompt and a control. Each control may be an instance of one of many different classes, including:

- a plain text window
- a choice list
- an action list of buttons (may only be the last item in the dialog box)
- a text edit box
- a numeric editor
- a floating point editor
- a time editor
- a date editor
- a scrolling or non-scrolling text editor
- a secret data input box
- a filename selector
- a filename editor
- an application-specific control

All these controls are subclasses of the LODGER class.

Any item in the dialog box may be separated from the following items by a horizontal line, referred to as an underline. In the case of the Series 3a and Work*about*, any number of items may be underlined, but for the Series 3, only one underline may appear in a dialog box. In the majority of dialogs an underline is used to separate the first item, designated to be the dialog title, from those that follow. The title is normally, but not necessarily, plain text.

Many dialogs can be created and run using the DLGBOX class directly, together with WSERV's `ws_do_dial` method.

Class definition

Defined in sub-category file *dlgbox.cl* (generated header file *dlgbox.g*).

```
CLASS   dlgbox          dlgchain
Dialog box class
    {
    REPLACE destroy
    REPLACE wn_key
    REPLACE wn_emphasise        Pass on to item with focus
    REPLACE wn_sense_help       Give start ID for help
    REPLACE wn_set              Set item by index
    REPLACE wn_sense            Sense item by index
    REPLACE wn_draw
```

```
ADD dl_item_lock            Lock an item by index
ADD dl_item_dim             Dim an item by index
ADD dl_set_item_flags       Set some dlgbox_item flags
ADD dl_set_prompt           Change the prompt for an item
ADD dl_take_focus           Move focus item specified by index
ADD dl_handle_to_index      Return index for given handle
ADD dl_index_to_handle      Return handle for given index
ADD dl_item_replace         Replace an existing item with another
ADD dl_item_append          Add an item (by rid) to end of list
ADD dl_init                 Initialise dialog from a resource file
ADD dl_dimmed_message       Present reason for item being dimmed
ADD dl_item_add             Add an item to the end of the list
ADD dl_set_size             Sets size of dialog after dynamic initialisation
ADD dl_inq_minsize=p_dummy  Allows default minimum sizes to be changed
ADD dl_dyn_init=p_dummy     For subclass dynamic initialisation
ADD dl_key                  Primarily for C call processing

DEFER dl_changed            For users that request item changed messages
DEFER dl_focus              For users that request change focus messages
DEFER dl_launch_sub         launch sub dialog if required
DEFER dl_item_new           Create instance of non-system dialog item

CONSTANTS
    {
    DLGBOX_NOTIFY_ENTER       0x0001  Notify when dialog terminated with enter
    DLGBOX_NOTIFY_ESCAPE      0x0002  Notify when dialog terminated with escape
    DLGBOX_RBUF_FILLED        0x0004  Set if subclasser fills rbuf
    DLGBOX_ACTION_LIST        0x0008  Dialog contains an action list
    DLGBOX_FROM_HWIF          0x0010  Reserved for system use
    DLGBOX_SMALL_FONT         DLGBOX_FROM_HWIF Dialog box uses the small font
    DLGBOX_NO_WAIT            0x0020  No am_start
    DLGBOX_NOTIFY_ALL_ACT     0x0040  Report ALL keys to dl_key
    DLGBOX_REPORT_ACT_HORIZ   0x0080  Report matching action key as horiz
    DLGBOX_APPEND_UNITS_TITLE 0x0100 Append units(cm/in) used to title
    DLGBOX_SMALL_ACTION_LIST  0x0200 Dialog contains a small action list
    DLGBOX_NO_SHADOW          0x0400 Create dialog with no shadow
    DLGBOX_NO_DDP             0x0800 Do not overwrite DatDialogPtr
    ! Four bits 0x1000 to 0x8000 reserved for PR_WIN_FORCE_XXX
    DLGBOX_ITEM_NOTIFY_CHANGED 0x0001 Report item changes to item
    DLGBOX_ITEM_DIMMED        0x0002  Item dimmed
    DLGBOX_ITEM_UNDERLINED    0x0004
    DLGBOX_ITEM_APPL_CAT      0x0008  Application specific item
    DLGBOX_ITEM_CENTRE        0x0010  Item centred in dialog
    DLGBOX_ITEM_DEAD          0x0020  Can't ever select item
    DLGBOX_ITEM_NOTIFY_FOCUS  0x0080  Report item focus changes to dialog
    DLGBOX_ITEM_NEEDS_PACK    0x0100  Add a pack selector immediately after
    DLGBOX_ITEM_LOCKED        0x0200  Visible & can take focus: values locked

    DLGBOX_ITEM_CAN_DEFER_X   0x0400  Can defer self-check
    DLGBOX_ITEM_X_PENDING     0x0800  Item must be checked before exit
    DLGBOX_ITEM_ACLIST        0x1000  This is the aclist in the dialog
    DLGBOX_LEFT_GAP           (BWIN_CUSHION_X+7)
    DLGBOX_BULLET_GAP         3
    DLGBOX_VERT_GAP           (BWIN_CUSHION_Y+2)
    UNDERLINE_DEPTH           3
    EXTRA_GUTTER_WIDTH        4
    DLGBOX_MAX_ITEM           9
    DLGBOX_ROMAN8_FONT        5       Add (WS_FONT_BASE-1) to get small font ID
    }
```

```
TYPES
    {
    typedef struct
        {
        UWORD flags;
        TEXT title[1];  ZTS string,
        ! followed by a byte count of the following items,
        ! each of which has a leading length byte
        } HD_DLGBOX_RSC;
    typedef struct
        {
        WORD flags;          initialisable DB_ITEM flags
        UBYTE class;         class of item
        TEXT prompt[1];      ZTS string
        ! followed by IN_<item> structure
        } AD_DLGBOX;         header for each item in resource information
    typedef struct
        {
        PR_LODGER *hand;       handle of item object
        PR_TEXTWIN *prompt;    handle of prompt for item
        WORD flags;            property DB_ITEM flags
        } DLGBOX_ITEM;         internal representation of items
    typedef union
        {
        DLGBOX_ITEM item[7];    Array of items in dialog
        DLGBOX_ITEM *pitm;      ptr to an array of more than 7 items
        } DLGBOX_ITEM_UNION;
    }

PROPERTY
    {
    DLGBOX_ITEM_UNION u;     either array of items or ptr to array of items
    VOID *rbuf;              address of result buffer
    WORD dimrid;             rid for dimmed item status message
    WORD helprid;            rid to start help
    UWORD flags;             holds DLGBOX_XXX flags
    UBYTE focus;             TRUE when an item has focus
    UBYTE count;             number of items (inc title) in dialog
    UBYTE current;           number of current control (0-6)
    UBYTE font;              LSB+1 of WS_FONT_BASE font ID
    UBYTE absorb;            direct all keypresses to the current control
    UBYTE changed;           TRUE if the current item needs self check
    }
}
```

On the Series 3 the CONSTANTS section of the class definition does not contain the definitions of:

```
DLGBOX_ITEM_ACLIST
EXTRA_GUTTER_WIDTH
DLGBOX_MAX_ITEM
```

but does contain two alternative definitions:

```
GUTTER_WIDTH              10
ACLIST_EXTRA_HEIGHT       20
```

In the PROPERTY section of the Series 3 class definition, the element:

```
DLGBOX_ITEM_UNION u;     either array of items or ptr to array of items
```

is replaced by the simpler:

```
DLGBOX_ITEM item[7];     array of items in dialog
```

These differences are associated with internal code changes and, apart from the additional number of items that a Series 3a dialog box may contain, have no significant consequences for the application programmer.

The Work*about* class definition introduces the defined constants DLGBOX_SMALL_FONT and DLGBOX_ROMAN8_FONT.

In the PROPERTY section of the Work*about* class definition, the line:

```
UBYTE font;              LSB+1 of WS_FONT_BASE font ID
```

has replaced the Series 3/3a line:

```
UBYTE underline;        y offset of S3 underline (not used in S3a code)
```

Property

`dlgbox.u.item` `dlgbox.u.pitm`	used to provide access to an array of `DLGBOX_ITEM` structs for the component items. If the dialog box contains seven or fewer items, the array is stored directly in `dlgbox.u.item`. For more than seven items, the array is held in allocated memory, pointed to by `dlgbox.u.pitm`. *In consequence, subclassers should not access the data in the array other than through the supplied methods, such as* `dl_index_to_handle`.
`dlgbox.rbuf`	the address of a user-supplied 'result' buffer, from which initialisation data may be read and to which result data may be written. The buffer's address may optionally be passed to the window server object's ws_do_dial method (or the equivalent `hLaunchDial` utility function).
`dlgbox.dimrid`	either zero or the resource ID of a text message that is to be displayed if a user attempts to modify a 'dimmed' control
`dlgbox.helprid`	either zero or the resource ID of a context-specific Help resource
`dlgbox.flags`	a collection of state flags, described below
`dlgbox.focus`	`TRUE` if keyboard focus is held by one of the dialog box controls. *Only system code may write to this item.*
`dlgbox.count`	a count of the number of items in the dialog box, including any dialog title. *Only system code may write to this item.*
`dlgbox.current`	the index of the dialog's component control that currently has focus. *Only system code may write to this item.*
`dlgbox.font`	introduced in the Work*about*, to replace `dlgbox.underline`. If not zero, its value determines the ID of the small font used in dialogs and, optionally, in list boxes. The font ID is given by `dlgbox.font+WS_FONT_BASE-1`. Although the value could, in principle, specify any font ID, it is restricted to be either zero or `DLGBOX_ROMAN8_FONT`. This is because the pseudo-static data held in `DatGate->gate.dx`, used to determine the size and position of dialog controls using the small font, is calculated for this font.
`dlgbox.underline`	not used in Series 3a code, and replaced by `dlgbox.font` in the Work*about*. On the Series 3 only, if not zero, it is the pixel offset from the top of the dialog box to a horizontal line drawn across the dialog box. On the Series 3, a dialog box may contain only one such line, normally used to separate the dialog title from the remainder of the dialog content.
`dlgbox.absorb`	`TRUE` if all keypresses are to be directed to the control that currently has the focus. Otherwise the dialog box intercepts keypresses that can be interpreted by any dialog action list, and those with keycodes `W_KEY_RETURN`, `W_KEY_ESCAPE`, `W_KEY_UP`, `W_KEY_DOWN`, `W_KEY_PAGE_UP` and `W_KEY_PAGE_DOWN`.
`dlgbox.changed`	set `TRUE` if the `lg_self_check` method of any of the dialog's component controls reports that its value has changed. Cleared when focus is transferred to another control.

DLGBOX flags

The content of `dlgbox.flags` may be any combination of the following flags:

`DLGBOX_NOTIFY_ENTER`	if this flag is set the dialog box will be sent a `DL_KEY` message when the dialog receives an Enter keypress
`DLGBOX_NOTIFY_ESCAPE`	if this flag is set the dialog box will be sent a `DL_KEY` message when the dialog receives an Esc keypress
`DLGBOX_RBUF_FILLED`	if this is not set and `dlgbox.rbuf` is not `NULL`, system code will write to `*dlgbox.rbuf` on exiting the dialog. A subclass that uses `dlgbox.rbuf` for its own purposes should set this flag
`DLGBOX_ACTION_LIST`	this flag must only be set if the dialog contains an action list (class `ACLIST`) as its last item. It is set automatically during the initialisation of an `ACLIST` component control. When set (provided `dlgbox.absorb` is `FALSE`) all received keys are first offered to the action list by sending it a `WN_KEY` message, which returns a value indicating whether or not the keypress matched one of the buttons
`DLGBOX_SMALL_ACTION_LIST`	this flag must only be set if the dialog contains a 'small' action list (class `SMACLIST`) as its last item. It is set automatically during the initialisation of an `SMACLIST` component control. When set (provided `dlgbox.absorb` is `FALSE`) all received keys are first offered to the action list by sending it a `WN_KEY` message, which returns a value indicating whether or not the keypress matched one of the buttons
`DLGBOX_NOTIFY_ALL_ACT`	if this flag is clear, report only those keys that match an action button by sending the dialog box a `DL_KEY` message, otherwise send a `DL_KEY` message for all keypresses that have been offered to the action list, irrespective of whether they matched an action button
`DLGBOX_REPORT_ACT_HORIZ`	if a keypress to an action button causes the dialog to terminate, a value may be written to `*dlgbox.rbuf` (see `DLGBOX_RBUF_FILLED`). If `DLGBOX_REPORT_ACT_HORIZ` is set, the value written to `*dlgbox.rbuf` is the index number of the button that matched the keypress (or -1 if it did not match). Otherwise the value is the uppercased key code of the keypress.
`DLGBOX_SMALL_FONT`	this flag is not defined for the Series 3 or 3a. If it is set in a dialog's resource, the dialog is initialised to use the small (8 point Roman) font, rather than the system font. This flag is not stored permanently in a dialog's property (see the `WSERV` `ws_do_dial` method)
`DLGBOX_NO_WAIT`	if this flag is clear, the dialog is run between `AM_START` and `AM_STOP` messages to the application manager
`DLGBOX_APPEND_UNITS_TITLE`	set this flag to append the current preferred units (cm or in) to any dialog title
`DLGBOX_NO_SHADOW`	if this flag is set the dialog box is created without a shadow border
`DLGBOX_NO_DDP`	if this flag is clear, the dialog box's handle is written to `DatDialogPtr` during initialisation and, on destruction, either `NULL` or the handle of the foremost of any remaining items in the `wserv.dial` list for which this flag is clear is written back to `DatDialogPtr`. Normally this flag is clear to improve the efficiency of dialog-related utility functions, which then do not need to pass the dialog box handle as a parameter
`PR_WIN_FORCE_RIGHT`	if set, the dialog will be positioned at the extreme right of the screen (see the `WIN` class)
`PR_WIN_FORCE_LEFT`	if set, the dialog will be positioned at the extreme left of the screen (see the `WIN` class)

PR_WIN_FORCE_BOTTOM	if set, the dialog will be positioned at the bottom of the screen (see the WIN class)
PR_WIN_FORCE_TOP	if set, the dialog will be positioned at the top of the screen (see the WIN class)

These flags are set, during creation of the dialog, from the initialisation data for the dialog box, which is normally loaded from a resource file.

Small font dialogs for the Work*about*

Work*about* dialogs may be set to use the small (8 point Roman) font by including the flag DLGBOX_SMALL_FONT in the flags field of the DIALOG resource that defines the dialog.

On initialisation of the dialog, this causes the value of dlgbox.font to be set to DLGBOX_ROMAN8_FONT and the value PR_WIN_IS_DLCTRL to be set in the win.flags property of each subclass of LODGER that is used as a component of the dialog.

All the supplied LODGER subclasses that may be used as dialog controls will draw themselves using the small font, but only if:

- they are components of a dialog i.e. if win.flags contains PR_WIN_IS_DLCTRL
- they are set to use the small font i.e. if ((PR_DLGBOX *)lodger->landlord)->dlgbox.font is non-zero

Outside a dialog box, or if the dialog is not set to use the small font, these subclasses of LODGER will draw themselves using the normal, system font.

When positioning controls within a dialog box, and when calculating the dimensions of controls and the dialog itself, the pre-calculated pseudo-static data is read from either DatGate->gate.x or DatGate->gate.dx, depending on whether the dialog is set to use the normal or the small font.

DLGBOX_ITEM flags

The DLGBOX_ITEM flags can be set individually for each dialog box item and are stored in the flags field of the corresponding DLGBOX_ITEM struct in the array that is either held in the dlgbox.u.item property, or pointed to by dlgbox.u.pitm. The content of the flags field may be any combination of the following flags:

DLGBOX_ITEM_NOTIFY_CHANGED	if this flag is set, changes to the item result in the dialog box receiving a DL_CHANGED message. This flag is normally only set in the item's initialisation data
DLGBOX_ITEM_NOTIFY_FOCUS	if this flag is set, each loss or gain of focus by the item results in the dialog box being sent a DL_FOCUS message. This flag is normally only be set in the item's initialisation data
DLGBOX_ITEM_UNDERLINED	if this flag is set, the corresponding item will be drawn with an underline extending across the full width of the dialog box. This flag is normally only set in the item's inintialisation data. For a Series 3 application it must not be set for more than one item in the dialog box
DLGBOX_ITEM_APPL_CAT	this flag should be set for any dialog item whose class definition is not in the HWIM category file. This flag may only be set in the item's initialisation data
DLGBOX_ITEM_CENTRE	this flag indicates that the corresponding item is to be centred in the dialog box. This flag is normally only set in the item's initialisation data
DLGBOX_ITEM_NEEDS_PACK	this flag indicates that, on initialisation of the dialog box, a pack selector control is to be added as the item immediately following this one. This flag may only be set in the item's initialisation data, and *must* be set for FNSELWIN or FNEDIT component controls
DLGBOX_ITEM_LOCKED	an item for which this flag is set is visible and can take focus, but its value may not be changed. This flag may be set or cleared by the dl_item_lock method

DLGBOX_ITEM_DIMMED	if this flag is set, the item is 'dimmed', that is, its control is made invisible. This flag may be set or cleared by the dl_item_dim method
DLGBOX_ITEM_DEAD	an item for which this flag is set may never take focus and may never be modified. This flag may only be set in the item's initialisation data
DLGBOX_ITEM_CAN_DEFER_X	an item for which this flag is set can, on losing focus, defer its self-check. A deferred check is automatically registered by the setting of the DLGBOX_ITEM_X_PENDING flag, described below. Applications will normally only set the DLGBOX_ITEM_CAN_DEFER_X flag in an item's initialisation data, but the flag is set automatically during initialisation of FNSELWN or FNEDIT component controls
DLGBOX_ITEM_X_PENDING	an item for which this flag is set must be checked before exiting the dialog. This flag is set and cleared by the system and should not be modified by application code
DLGBOX_ITEM_ACLIST	this flag indicates that the corresponding item has a control that is an instance of either ACLIST or SMACLIST. This flag is set by the system and should not be present in any dialog resource, nor should it be modified by application code

DLGBOX methods

In addition to optionally providing replacements for the DEFERred methods:

```
dl_changed
dl_focus
dl_launch_sub
dl_item_new
```

applications are, in general, not expected to subclass methods other than:

```
dl_dyn_init
dl_key
dl_inq_minsize
dl_set_size
```

Occasionally a dialog box may additionally need to subclass one or more of:

```
dl_item_add
dl_dimmed_message
wn_sense_help
```

Consistency checks

A dialog box control may need to perform a consistency check on its content, a typical case being a numeric control whose value must remain within prescribed limits.

In simple cases a dialog may check the consistency of its data in its dl_key method. However, in many cases it is either not feasible or inappropriate to check the value at each keypress that modifies the value. For example, a numeric edit box whose value is constrained to lie between 10 and 50 may transiently contain the 'illegal' text "1" while the number "18" is being typed in.

The DLGBOX class provides a mechanism for checking the validity of the content of its controls, triggered either when focus is transferred to one of its controls, or when the dialog is being terminated by any means other than by pressing Esc. Since a control's validity may, for example, depend on the values of one or more of the other controls, the check on change of focus may be deferred until the dialog termination, when the final values of all controls are known. Such items are indicated by setting the DLGBOX_ITEM_CAN_DEFER_X flag in the flags field of the corresponding DLGBOX_ITEM struct.

The check on an item is always deemed to succeed if the item is locked or dimmed, or if the check is deferred. Otherwise the control is sent an LG_SELF_CHECK message (see the description of this method for the LODGER class, described in the *Windows* chapter). This message returns a value that indicates whether or not the content has changed as well as whether the check failed or succeeded.

Regardless of the success or failure reported by the lg_self_check method, if the return value indicates that the value has changed, dlgbox.changed is set to TRUE and, provided the flags field of the relevant DLGBOX_ITEM struct contains DLGBOX_ITEM_NOTIFY_CHANGED, the dialog box is sent a DL_CHANGED message.

On any failure the checking stops and focus is set to the item that failed its check. This may prevent the user from moving to another item in the dialog or from exiting the dialog (except by pressing Esc to cancel the dialog) until the item is modified so that it passes the check.

DL_INIT — Initialise

```
INT dl_init(HD_DLGBOX_RSC *head, VOID *rbuf);
```

Initialise a dialog box from an item in a resource file. This method is called from the WSERV ws_do_dial method to create and initialise the component controls (before calls to the dl_dyn_init, and dl_set_size methods).

The method performs the following actions:

- copies the value of rbuf, which points to a user-supplied 'result' buffer, into dlgbox.rbuf, and copies head->flags into dlgbox.flags
- sets win.flags to PR_BWIN_CUSHION ORed with PR_BWIN_CORNER_4 and, if dlgbox.flags does not contain DLGBOX_NO_SHADOW, ORS PR_BWIN_SHADOW_2 into win.flags
- if head->flags contains DLGBOX_NO_DDP, ORS PR_WIN_NO_DDP into win.flags, otherwise sets the magic static DatDialogPtr to the handle of the dialog box
- sends itself a WN_CONNECT message, with a parent pointer of NULL to create a top-level window, and with a zero flags field parameter so that the window is initially of indeterminate size and position
- if head->title does not contain a null string, creates and initialises an instance of TEXTWIN containing this text, with centred alignment. This is made the first, unprompted, item in the dialog box:
  its handle is written to the hand element of the dialog's first DLGBOX_ITEM struct
  the flags element of this struct is set to DLGBOX_ITEM_DEAD|DLGBOX_ITEM_CENTRE
  the prompt element of this struct is left NULL
  the title is set be underlined
  dlgbox.count is set to 1
  On the Work*about*, the initialisation of this instance of TEXTWIN is preceded by setting the PR_WIN_IS_DLCTRL flag in the instance's win.flags. This allows the text window to determine whether it needs to draw itself using the smaller font.
- the byte following the zero terminator of any string in head->title is read to determine the number of items in any following list of items to be included in the dialog (this byte is generated by the resource compiler). The specified items are added to the dialog box, in order, with a sequence of DL_ITEM_ADD messages

Any failure to add an item results in p_leave being called.

The method returns zero to indicate that no failure has occurred. It is suitable for calling under the protection of p_enter.

This method is not intended to be replaced.

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the dialog box and its component items.

The method first sends DESTROY messages to each of the prompts and controls whose handles are stored in the dialog box's array of DLGBOX_ITEM structs.

Further action depends on how the dialog box was started up:

- if the start-up of the dialog by the WSERV ws_do_dial method completed without error (this is indicated by the dialogs win.flags containing PR_WIN_INITIALISED) the dialog box will have

been added to the wserv.dial list. In such a case w_ws is sent a WS_REMOVE_DIAL message, passing the dialog's handle

- the DatDialogPtr magic static may contain the dialog's handle. If this is so, the wserv.dial list is searched for the foremost item that does not have PR_WIN_NO_DDP set in its win.flags and DatDialogPtr is set to contain this item's handle, or NULL if no such item is found
- if the dialog was successfully started by ws_do_dial and does not have DLGBOX_NO_WAIT set in dlgbox.flags, an AM_START message was sent to w_am, so the destroy method sends w_am an AM_STOP message

In all cases, before the optional sending of the AM_STOP message, the method supersends the DESTROY message.

This method is not intended to be replaced.

DL_ITEM_ADD — Add an item

```
INT dl_item_add(AD_DLGBOX *par);
```

Append, as the current last line of the dialog box, the control specified by the AD_DLGBOX struct pointed to by par. This struct is defined in *dlgbox.g* as:

```
typedef struct
    {
    WORD flags;         /* initialisable DB_ITEM flags */
    UBYTE class;        /* class of item */
    TEXT prompt[1];     /* ZTS string */
    /* followed by an IN_XXX component-specific initialisation structure */
    } AD_DLGBOX;        /* header for each item in resource information */
```

It is a programming error, with unpredictable results, to add an item if the dialog box currently contains the maximum number of items.

The method first creates an instance of the class par->class, writing its handle to the hand element of the corresponding DLGBOX_ITEM struct. If par->flags does not contain DLGBOX_ITEM_APPL_CAT the class is assumed to be a standard control in the HWIM category. Otherwise the method sends a DL_ITEM_NEW message to create the instance. The par->flags value is copied into the flags element of the corresponding DLGBOX_ITEM struct.

If the string at the start of the par->prompt buffer is not a null string, an instance of TEXTWIN is created, writing its handle to the prompt element of the corresponding DLGBOX_ITEM struct. This instance is sent a WN_INIT message, passing the address of an IN_TEXTWIN struct and the dialog box handle. The IN_TEXTWIN struct contains the prompt string and a state value of 0 if par->flags contains any of DLGBOX_ITEM_DIMMED, DLGBOX_ITEM_LOCKED or DLGBOX_ITEM_DEAD, otherwise state is set to IN_TEXTWIN_BULLET.

The method then sends a WN_INIT message to the item's control, passing a pointer to any data (assumed to be a suitable initialisation structure) that follows the terminating zero of the string in the par->prompt buffer. Further parameters are the dialog box handle and the handle of the control in the previous line[1] (see also the *Special note* below).

On the Work*about*, the sending of the WN_INIT message to both the control and any prompt is preceded by setting the PR_WIN_IS_DLCTRL flag in the corresponding object's win.flags. This allows the control or prompt to determine whether it needs to draw itself using the smaller font.

If par->flags contains DLGBOX_ITEM_UNDERLINED, the item will be drawn with an underline (only one underline is allowed on the Series 3).

Finally, the value of dlgbox.count is incremented by one.

The method returns zero to indicate that no failure has occurred. It is thus suitable for calling under the protection of p_enter.

---

[1]This will contain an indeterminate value when adding the first item, but the wn_init methods of most controls ignore this value. Those that make use of this parameter are never used as the first item in the dialog box.

This method may be either used or replaced by application writers. It is called by system code, once for each component control, from the WSERV ws_do_dial method after the dialog resource has been loaded from the resource file.

An application may call this method to add one or more items, depending on run-time circumstances, although the dl_item_append method - which also loads the item's resource - will usually be more appropriate. All such uses must be before the DL_SET_SIZE message is processed at the DLGBOX level. Application code will typically send a DL_ITEM_ADD message from the dl_dyn_init method.

Since the method is called by system code once for each item that appears in the dialog resource, it may be replaced to omit one or more items, or to add items at positions other than the end of the dialog, depending on run-time circumstances.

The following example is a replacement of the dl_item_add method that optionally omits the third item (with index number 2) from a dialog. It assumes that the MYDLG subclass of DLGBOX adds two items of property: mydlg.needed, which is TRUE if the item is to be included, and mydlg.omitted which is initially FALSE and is set to TRUE if the item is not included in the dialog (an alternative would be to store this information in a result buffer, pointed to by dlgbox.rbuf).

```
METHOD VOID mydlg_dl_item_add(PR_MYDLG *self, AD_DLGBOX *par)
    {
    if ((self->dlgbox.count==2)&&(!self->mydlg.needed)&&(!self->mydlg.omitted))
        self->mydlg.omitted=TRUE;
    else
        p_supersend3(self,O_DL_ITEM_ADD,par);
    }
```

Special note

If par->flags contains DLGBOX_ITEM_NEEDS_PACK (normally only if adding a control of either the FNEDIT or the FNSELWN class) then this method will result in the addition of *two* controls, thus occupying two lines in the dialog box. After the creation of the first control, but before its initialisation, the method sends a further, recursive, DL_ITEM_ADD message to create a PACKSEL control with an associated "Disk" prompt string (this prompt is read from the SYS_PACK resource in the system resource file). In this case the final parameter passed in the WN_INIT message to the first control is the handle of the *following* control (that is, the instance of PACKSEL). If par->flags contains DLGBOX_ITEM_UNDERLINED, the underline is transferred to be below the PACKSEL control, so that an item and its corresponding pack selector will never be separated by an underline.

DL_ITEM_APPEND — Add an item by resource ID

```
VOID dl_item_append(INT rid);
```

Append the item specified by the resource item with resource ID rid.

The resource item is loaded into a temporarily allocated buffer, the loaded data being assumed to be an AD_DLGBOX struct. The control and any associated prompt are appended to the dialog box, as described for the dl_item_add method. The temporarily allocated buffer is freed before the method returns.

The method must not be used to add an item following any action list item.

An application may call this method to add one or more items at the bottom of a dialog, depending on run-time circumstances, for example, to generate two different, but similar, dialogs from a common dialog box resource. All such uses must be before the DL_SET_SIZE message is processed at the DLGBOX level. A call from application code to the dl_item_append method will typically be from the dl_dyn_init method.

This method is not intended to be replaced.

DL_ITEM_REPLACE — Replace an existing item

```
VOID dl_item_replace(INT index, INT rid);
```

Replace existing item number index by the item specified by the resource item with resource ID rid.

The existing control and any corresponding prompt are sent DESTROY messages. The resource item is then loaded into a temporarily allocated buffer, the loaded data being assumed to be an AD_DLGBOX struct. The replacement item and any corresponding prompt are created and initialised as described for the

dl_item_add method, their handles overwriting those that they replace. The method does not increment dlgbox.count. The temporarily allocated buffer is freed before the method returns.

This method must not be used with a resource which has the DLGBOX_ITEM_NEEDS_PACK flag set in its flags data.

An application may call this method to replace one or more items, depending on run-time circumstances, for example, to generate two different, but similar, dialogs from a common dialog box resource. All such uses must be before the DL_SET_SIZE message is processed at the DLGBOX level. A call from application code to the dl_item_replace method will typically be from the dl_dyn_init method.

This method is not intended to be replaced.

WN_SET — Set item by index

```
VOID wn_set(INT index, VOID *par);
```

Set one or more data elements in the property of the control associated with the dialog box item with index number index, by sending a WN_SET message to the control.

The parameter par is assumed to be a pointer to a struct that specifies the data to be set. The type of struct that is expected depends on the class of the control that is being set; the various structs are described in the following *Dialog Controls* chapter.

This method will normally be used (rather than replaced) by application writers. See also the various hDlgSetXXX utility functions.

DL_SET_PROMPT — Change the prompt for an item

```
VOID dl_set_prompt(INT index, SE_TEXTWIN *par);
```

Send a WN_SET message to the prompt associated with item number index, passing the parameter par, assumed to be a pointer to an SE_TEXTWIN struct.

This method replaces an existing prompt; it can not be used to add a prompt to an item that does not already possess a prompt. Thus the method may not be used on an item for which no prompt was defined in the item's resource.

It is perfectly permissible for a replacement prompt to be longer than the prompt text that was supplied in the dialog's resource. However, such a replacement may alter the width needed to display the dialog. If this is the case, the replacement should not be made after the WSERV ws_do_dial method has sent the dialog a DL_SET_SIZE message.

This method is not intended to be replaced.

WN_SENSE — Sense item by index

```
VOID wn_sense(INT index, VOID *par);
```

Sense the property of the control associated with the dialog box item with index number index, by sending a WN_SENSE message to the control.

The parameter par is assumed to be a pointer to a struct that matches the data to be sensed. The type of struct that is expected depends on the class of the control that is being sensed; the various structs are described in the following *Dialog Controls* chapter.

This method will normally be used (rather than replaced) by application writers. See also the various hDlgSenseXXX utility functions.

WN_KEY — Handle a keypress

```
INT wn_key(INT keycode, INT modifiers);
```

The description of this method is included for interest only. Applications should neither replace this method nor call it explicitly.

Handle a keypress, directing it, if necessary, to a component control. This message is sent by the application's instance of WSERV. Any non-zero return value from the wn_key method will terminate the dialog, causing the dialog box to receive a DESTROY message.

If dlgbox.absorb is TRUE, all keys are directed to the current control, as described later. Otherwise keys are processed as follows.

If the dialog contains an action list (which must always be the last item) the action list is sent a WN_KEY message, passing a key code of p_toupper(keycode). This message returns either a negative value if the keypress does not match any of the action buttons, or the index number of the button that matched the keypress (the leftmost button has index number zero).

If there was no match and dlgbox.flags does not contain DLGBOX_NOTIFY_ALL_ACT , processing continues with the testing for specific keys, as described later.

Otherwise (that is, either the keypress matches an action button, or dlgbox.flags contains DLGBOX_NOTIFY_ALL_ACT) a forced consistency check is applied to all controls. If any control fails its check the method terminates, returning WN_KEY_NO_CHANGE (0) so that the dialog box is not exited. If the consistency check succeeeds, the method sends a DL_KEY message, passing dlgbox.current, p_toupper(keycode) and the key-matching value returned by the WN_KEY message that was sent to the action list. The method terminates, returning the value returned by the DL_KEY message. Before returning, and provided that:

- the return value is non-zero (that is, the dialog is terminating)
- dlgbox.rbuf is not NULL
- dlgbox.flags does not contain DLGBOX_RBUF_FILLED

a WORD of data is written to *dlgbox.rbuf. If dlgbox.flags contains DLGBOX_REPORT_ACT_HORIZ this data is the index number of the action button that was selected. Otherwise the value written to *dlgbox.rbuf is the uppercased key code.

Provided the keypress has not been processed by an action list, the following specific keypress codes are tested:

W_KEY_RETURN — this key is ignored if the dialog contains an action list (dlgbox.flags contains DLGBOX_ACTION_LIST or DLGBOX_SMALL_ACTION_LIST) in which case the method terminates immediately, returning WN_KEY_NO_CHANGE. Otherwise a forced consistency check is applied to all controls. If any control fails its check the method terminates, returning WN_KEY_NO_CHANGE.

If dlgbox.flags does not contain DLGBOX_NOTIFY_ENTER the method just returns WN_KEY_CHANGED, otherwise it returns the result from sending a DL_KEY message. In either case, provided:

- the return value is non-zero (that is, the dialog is terminating)
- dlgbox.rbuf is not NULL
- dlgbox.flags does not contain DLGBOX_RBUF_FILLED

the value of dlgbox.current is written to the WORD pointed to by dlgbox.rbuf.

W_KEY_ESCAPE — provided dlgbox.rbuf is not NULL and dlgbox.flags does not contain DLGBOX_RBUF_FILLED write, to the WORD pointed to by dlgbox.rbuf, either W_KEY_ESCAPE or, if dlgbox.flags contains DLGBOX_REPORT_ACT_HORIZ, a value of -1

W_KEY_UP	move up 'one' item in the dialog. The current item becomes the first earlier item whose flags do not contain DLGBOX_ITEM_DEAD. A DL_TAKE_FOCUS message is sent, passing the new current item's index number. Moving up from the first item in the dialog positions to the last item
W_KEY_DOWN	move down 'one' item in the dialog. The current item becomes the first following item whose flags do not contain DLGBOX_ITEM_DEAD. A DL_TAKE_FOCUS message is sent, passing the new current item's index number. Moving down from the last item in the dialog positions to the first item
W_KEY_PAGE_UP	move to the 'first' item in the dialog. The current item becomes the first item whose flags do not contain DLGBOX_ITEM_DEAD. A DL_TAKE_FOCUS message is sent, passing the new current item's index number.
W_KEY_PAGE_DOWN	move to the 'last' item in the dialog. The current item becomes the last item whose flags do not contain DLGBOX_ITEM_DEAD. A DL_TAKE_FOCUS message is sent, passing the new current item's index number.

If dlgbox.focus is FALSE, any other keypress is ignored and the method returns WN_KEY_NO_CHANGE. Otherwise the keypress is directed to the dialog's current component control, as explained in the following paragraphs. All incoming keys are processed in this way if dlgbox.absorb is TRUE.

If the flags field of the current item contains DLGBOX_ITEM_DIMMED or DLGBOX_ITEM_LOCKED, a DL_DIMMED message is sent, to inform the user that the item can not be modified, and the method then returns WN_KEY_NO_CHANGE.

Otherwise the current control is sent a WN_KEY message, and further processing depends on the return value:

- if the return value is WN_KEY_ABSORB_ON, dlgbox.absorb is set TRUE, so that all subsequent keys will be directed to the current control
- if the return value is anything other than WN_KEY_NO_CHANGE and dlgbox.absorb is TRUE, dlgbox.absorb is set FALSE, so that subsequent keys may be processed by the dialog box as described above
- if the return value is WN_KEY_CHANGED and the current item's flags contain DLGBOX_ITEM_NOTIFY_CHANGED, the dialog box is sent a DL_CHANGED message

In all these cases the dialog wn_key method returns WN_KEY_NO_CHANGE.

DL_KEY — Handle key input

```
INT dl_key(INT index, INT keycode, INT actbut);
```

Process a keypress that may potentially exit the dialog.

This method is supplied so that it may be replaced to perform application-specific processing of such keypresses.

This message is sent by the dialog box wn_key method in the following circumstances:

- when dlgbox.flags contains DLGBOX_NOTIFY_ENTER and the dialog is about to terminate after receiving a W_KEY_RETURN. The value of keycode is W_KEY_RETURN and actbut is -1
- when dlgbox.flags contains DLGBOX_NOTIFY_ESCAPE and the dialog is about to terminate after receiving a W_KEY_ESCAPE. The value of keycode is W_KEY_ESCAPE and actbut is -1
- the dialog is about to terminate after receiving a key that matches a button in an action list. The value of keycode is the uppercased key code that was passed to the action list's wn_key method and actbut is the index (0 for the leftmost button) of the matching button
- following any non-matching keypress received by an action list, provided dlgbox.flags contains DLGBOX_NOTIFY_ALL_ACT. The value of keycode is the uppercased key code that was passed to the action list's wn_key method and actbut is -1

In all cases index contains the current value of dlgbox.current.

The method should return WN_KEY_NO_CHANGE to prevent termination of the dialog, or WN_KEY_CHANGED to confirm termination (which causes the dialog box to be sent a DESTROY message). Before returning

WN_KEY_CHANGED the method is responsible for ensuring that any relevant dialog box state is either saved or returned to the initiator of the dialog box. Data may conveniently be returned to the initiator via a user-supplied buffer pointed to by dlgbox.rbuf.

The supplied method simply returns WN_KEY_CHANGED.

DL_TAKE_FOCUS — Move focus to specified item

```
INT dl_take_focus(INT index);
```

Attempt to change the focus to item number index.

A deferred consistency check is applied to the current item. If the check fails the focus change is not made. Otherwise, provided the specified item is not already the one with focus, the focus is switched to the specified item:

- any prompt of the item that is losing focus is sent a WN_EMPHASISE,FALSE message
- provided its flags do not contain DLGBOX_ITEM_DIMMED or DLGBOX_ITEM_LOCKED, the control of the item that is losing focus is also sent a WN_EMPHASISE,FALSE message
- if the flags of the item that is losing focus contain DLGBOX_ITEM_NOTIFY_FOCUS, the dialog box is sent a DL_FOCUS message to inform it that the item is losing focus
- a TRUE value is written to dlgbox.focus and dlgbox.current is set to index, so that it now refers to the item that is gaining focus
- any prompt of the item that is gaining focus is sent a WN_EMPHASISE,TRUE message
- provided the item that is gaining does not have the either of the flags DLGBOX_ITEM_DIMMED or DLGBOX_ITEM_LOCKED set, the control of this item is also sent a WN_EMPHASISE,TRUE message
- if the item that is gaining focus has the flag DLGBOX_ITEM_NOTIFY_FOCUS set, the dialog box is sent a DL_FOCUS message to inform it that the new item is gaining focus

The method returns FALSE if a failed consistency check prevents a change of focus being made. In all other circumstances (including the case where the specified item is already the one with focus) the method returns TRUE.

This method is not intended to be replaced. It is not suitable for being called from the dl_dyn_init method. If an application wishes to set the focus on initialisation, it should do so from a replaced dl_set_size method. It may be called from any other method (such as dl_key) once the dialog has been made visible.

WN_DRAW — Draw whole dialog

```
VOID wn_draw(VOID);
```

Draw the dialog box and its contents.

The method first draws the dialog box border. It then sends a WN_DRAW message to every prompt, and to every control that does not have DLGBOX_ITEM_DIMMED set in the flags field of the corresponding DLGBOX_ITEM struct. If the flags field of a control also has DLGBOX_ITEM_UNDERLINED bit set, a horizontal line is drawn below the item, extending across the full width of the area inside the window's border.

On the Series 3, only one underline can be present in a dialog box and its vertical position is specified by dlgbox.underline. If this value is non-zero, a horizontal line is drawn at the specified position, extending across the full width of the area inside the window's border.

On the Work*about*, all text is drawn in the small (8 point Roman) font and line heights are calculated accordingly.

DL_SET_SIZE — Set size of dialog

```
INT dl_set_size(VOID);
```

Set the height and width of the dialog to the values required to display all the component items, and set the position all components within the dialog box. An LG_SENSE_WIDTH message is sent to each component to determine the width needed to display it.

A DL_SET_SIZE message is sent from the WSERV ws_do_dial method, after that method has sent the dialog box DL_INIT and DL_DYN_INIT messages.

If dlgbox.flags contains DLGBOX_APPEND_UNITS_TITLE the SYS_CENTIMETRES or SYS_INCHES resource (depending on whether w_ws->wserv.flags contains PR_WSERV_METRIC) is loaded from the system resource file and appended to the dialog title. This will call p_panic if the dialog has no title.

Some dialog items (such as an instance of TEXTWIN used as a title) are composed of a single control component, whereas others are formed from two components - a prompt and a control. The width of each component is found by sending it an LG_SENSE_WIDTH message. The dialog box width is set to contain the widest item from each of these two groups, subject to any explicit minimum widths set by sending each control a DL_INQ_MINSIZE message.

The width for single-component items is the larger of the width written to *pOverallWidth by dl_inq_minsize (zero by default) and the single-component item of greatest width.

The width for two-component items is the width of the widest prompt plus the width of the widest control (respectively not less than any values written to *pPromptWidth and *pControlWidth by dl_inq_minsize) plus a minimum gutter width separation between them. Provided at least one item is not marked with the DLGBOX_ITEM_DEAD flag, the prompt width allows space for the prompt to include a leading bullet to show that an item can be modified.

The width of the dialog box is the greater of these two widths plus an allowance for the dialog box borders and a small gap at either side. If this is wider than the screen, an attempt is made to clip the right hand side of the control components. On all machines except the Work*about*, if such clipping means that one or more controls will be entirely invisible the method calls p_leave(E_GEN_TOOWIDE).

The height of the dialog box is just the sum of the heights of the controls, plus the top and bottom borders and a small additional amount for each item that is underlined. On all machines except the Work*about*, if the total exceeds the screen height the method calls p_leave(E_GEN_TOOMANY). In most cases there will always be room for the maximum number of items, but some items (for example, instances of ACLIST) take up additional height.

The method then positions each prompt and control by sending it an LG_SET_ID_POS message. Unless marked as centred, prompts are positioned at the left side, aligned at their left edges, and controls occupy a right hand region, again aligned at their left edges. The prompt and control regions are aligned with the edges of the widest centred control, subject to their being separated by a small gap.

Finally, provided that not all items are marked with the DLGBOX_ITEM_DEAD flag, the method sets the focus to the first such unmarked item - dlgbox.focus is set TRUE, dlgbox.current is set to the item and a WN_EMPHASISE,TRUE message is sent to the item's prompt, if it exists. The control itself is not sent a WN_EMPHASISE message as it will receive one later, when the dialog box itself receives a WN_EMPHASISE message.

The method returns zero to indicate that p_leave has not been called. It is suitable for calling under the protection of p_enter.

This method may be replaced, but should not be called explicitly from application code. It offers the last opportunity to modify the dialog box content before it becomes visible. A common use is to modify the content after the size of the dialog box has been calculated.

Any replacement method should only add further processing, before and/or after supersending the DL_SET_SIZE message.

DL_ITEM_DIM — Dim an item

```
VOID dl_item_dim(INT index, UINT flag);
```

Undim item number `index` if `flag` is FALSE, otherwise dim it. When an item is dimmed its control is not displayed and its prompt does not display a bullet point. It is harmless to send a dimmed item a DL_ITEM_DIM, TRUE message, or an undimmed item a DL_ITEM_DIM, FALSE message.

Dimming the item clears the DLGBOX_ITEM_DIMMED flag in item number `index` and sends any prompt (an instance of TEXTWIN) a WN_SET message to clear its PR_TEXTWIN_BULLET flag. The control is sent a WN_VISIBLE, FALSE message.

Undimming the item reverses these flag changes and sends the control a WN_VISIBLE, TRUE message.

In either case the WN_VISIBLE message is sent only if the dialog box `win.flags` contains PR_WIN_INITIALISED. This prevents any control from drawing itself during the initialisation of the dialog box, before the whole dialog box can be made visible.

If item number `index` has the DLGBOX_ITEM_NEEDS_PACK flag set, the dimming/undimming operation is also performed on the following PACKSEL item.

This method is not intended to be replaced.

DL_ITEM_LOCK — Lock an item

```
VOID dl_item_lock(INT index, UINT flag);
```

Unlock item number `index` if `flag` is FALSE, otherwise lock it. When an item is locked its prompt does not display a bullet point but, unlike a dimmed item, its control remains visible. It is harmless to send a locked item a DL_ITEM_LOCK, TRUE message, or an unlocked item a DL_ITEM_LOCK, FALSE message.

Locking the item clears the DLGBOX_ITEM_LOCKED flag in the item's flags field and sends any prompt (an instance of TEXTWIN) a WN_SET message to clear its PR_TEXTWIN_BULLET flag.

Unlocking the item reverses these flag changes.

If item number `index` has the DLGBOX_ITEM_NEEDS_PACK flag set, the locking/unlocking operation is also performed on the following PACKSEL item.

This method is not intended to be replaced.

DL_DIMMED_MESSAGE — Display 'dimmed' message

```
VOID dl_dimmed_message(VOID);
```

Display a status message, using `hInfoPrint`, indicating that a locked or dimmed item can not be modified. This method will be called by system code when the uset attempts to modify an item that is either dimmed or locked.

The resource ID passed to `hInfoPrint` is `dlgbox.dimrid` or, if this is zero, either of the system resource ID's SYS_DIMMED_MSG or SYS_LOCKED_MSG depending on whether or not the item's flags contain DLGBOX_ITEM_DIMMED (if this flag is not present the item is assumed to be locked, without testing for the presence of DLGBOX_ITEM_LOCKED).

This method can be replaced, for example, to display a context-sensitive message.

DL_HANDLE_TO_INDEX — Sense item index

```
INT dl_handle_to_index(PR_LODGER *handle);
```

Return the index number for the item whose control handle is `handle`.

Returns -1 if `handle` does not match any dialog item.

This method is not intended to be replaced.

DL_INDEX_TO_HANDLE — Sense item handle

```
PR_LODGER *dl_index_to_handle(INT index);
```

Return the handle of the control in item number `index`.

It is a programming error to call this method with a value of `index` that does not correspond to an existing dialog box item.

This method is not intended to be replaced.

DL_SET_ITEM_FLAGS — Set item flags

```
VOID dl_set_item_flags(PR_LODGER *lodger, UINT flags);
```

OR the passed `flags` into the flags field of the dialog box item corresponding to the control with handle `lodger`.

This method is intended for use by a component control to set some aspect of the dialog's state. It is used, for example, by the FNSELWN and ACLIST dialog control classes.

The DLGBOX_ITEM_LOCKED and DLGBOX_ITEM_DIMMED flags should be set (or cleared) by use of the appropriate specific method. Note that, apart from these two flags, no means is provided for clearing any of an item's flags.

This method is not intended to be replaced.

WN_SENSE_HELP — Sense start ID for Help

```
INT wn_sense_help(PR_DLGBOX *self);
```

The action depends on the value of `dlgbox.helprid` as follows:

- return `dlgbox.helprid`, if it is greater than zero.
- if `dlgbox.helprid` is zero, return the result of supersending the WN_SENSE_HELP message. This is handled by the WIN superclass and returns either `w_ws->wserv.help_index_id` or, if this is zero, the (negative) system resource ID `-SYS_HELP_ON_HELP`.
- if `dlgbox.helprid` is -1, call `p_leave(RUN_ACTIVE_USED)`. This is used by the ERRORDLG class, to disable the display of help information when an error is being reported (so that, for example, multiple nested out of memory errors can not be generated by requesting Help while an out of memory error report is visible).

WN_EMPHASISE — Set dialog box emphasis

```
VOID wn_emphasise(UINT flag);
```

Emphasise the dialog box if flag is TRUE, otherwise de-emphasise it.

First supersends the WN_EMPHASISE message to set the border emphasis. If an item has the keyboard focus (`dlgbox.focus` is TRUE) and if the current item is not dimmed or locked, send a WN_EMPHASISE message to the control of this item.

DL_INQ_MINSIZE — Specify minimum widths

```
VOID dl_inq_minsize(INT *pOverallWidth, INT *pPromptWidth, INT *pControlWidth);
```

Inform the caller of the minimum widths for whole line items (`*pOverallWidth`) and for the prompt (`*pPromptWidth`) and control (`*pControlWidth`) segments of two-part items. The method is called from the `dl_set_size` method and, on entry, all three parameters point to locations containing zero.

The supplied method does nothing.

A subclass may replace this method to write application-specific minimum values to any or all of `*pOverallWidth`, `*pPromptWidth` or `*pControlWidth`. On all machines except the Work*about*, writing

values that force a dialog to exceed the width of the screen will cause `dl_set_size` to call `p_leave(E_GEN_TOOWIDE)`.

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

A `DL_DYN_INIT` message is sent from the `WSERV` `ws_do_dial` method, after that method has sent the dialog box a `DL_INIT` message, but before the sending of a `DL_SET_SIZE` message.

The supplied method does nothing. It is supplied so that it may be replaced to perform application-specific initialisation of the dialog box after all its items have been added, but before its size is calculated and it is made visible.

The most common use for this method is to set the initial value of one or more controls, depending on the current state of the application, but other actions may include dimming, locking, adding or replacing one or more dialog items. Initialisation data may conveniently be passed by means of a buffer pointed to by `dlgbox.rbuf`.

Deferred DLGBOX methods

There is no general requirement to subclass `DLGBOX` to provide any of these `DEFER`red methods. Each method need be supplied only if the conditions are satisfied for the corresponding message to be received.

DL_CHANGED — Item changed message

```
VOID dl_changed(INT index);
```

Notify the dialog box that the item with index number `index` has changed in some way.

This message will only be received for values of `index` for which the flags field of the corresponding item contains `DLGBOX_ITEM_NOTIFY_CHANGED`. The method need not be supplied for any dialog in which no items are so marked.

A typical use would be to modify an item - say the allowed range of a numeric edit box - as a result of changes made by the user to some other item.

DL_FOCUS — Focus changed message

```
VOID dl_focus(WORD index, WORD flag);
```

Notify the dialog box that the item with index number `index` has gained or lost focus. The value of flag is `TRUE` if the item has gained focus, or `FALSE` if the item has lost focus.

This message will only be received for values of `index` for which the flags field of the corresponding item contains `DLGBOX_ITEM_NOTIFY_FOCUS`. The method need not be supplied for any dialog in which no items are so marked.

A common use is to detect when an edit box loses focus. This may be an appropriate time to sense the value and make any necessary modifications to other dialog box items.

DL_LAUNCH_SUB — Launch sub-dialog if required

```
VOID dl_launch_sub(INT index);
```

This message is sent by the window server object if, on return from a `wn_key` message sent to the dialog, the value of `w_ws->wserv.subdial` is non-zero. The value of `index` is the item number of the item that is launching the subdialog.

The method need not be supplied for any dialog that does not write a non-zero value to `w_ws->wserv.subdial`. For further information, see the description of the use of subdialogs in the *Dialogs* chapter of the *Object Oriented Programming Guide*.

DL_ITEM_NEW — Create non-system dialog item

```
VOID *dl_item_new(AD_DLGBOX *par);
```

Create an instance of the application-specific dialog item with class number `par->class`.

Assuming that the application category file is *myapp.cat*, the code of a `dl_item_new` method for a dialog box that has application-specific items defined only in this category would be:

```
f_new(CAT_MYAPP_MYAPP,par->class);
```

If, exceptionally, a dialog box contains two or more application-specific items, with classes defined in different categories, the method will need additional logic to create the item from the appropriate category.

This message will only be received if the flags field associated with one or more of the items in a dialog box contains DLGBOX_ITEM_APPL_CAT, indicating that the item does not have its class definition in the HWIM category. The method need not be supplied for any dialog whose items are not so marked.

ATSDIAL

WIN	BWIN	DLGCHAIN	DLGBOX		ATSDIAL
flags		next	item	count	
id			rbuf	current	
			dimrid	underline	
			helprid	absorb	
			flags	changed	
			focus		
~~destroy~~	~~wn_draw~~		~~destroy~~	dl_item_replace	destroy
wn_calc_position	~~wn_emphasise~~		wn_key	dl_item_append	dl_item_add
wn_connect			wn_emphasise	dl_init	dl_dyn_init
wn_dodraw			wn_sense_help	dl_dimmed_message	dl_key
~~wn_emphasise~~			wn_set	~~dl_item_add~~	
~~wn_key~~			wn_sense	dl_set_size	
wn_position			wn_draw	dl_inq_minsize	
wn_redraw			dl_item_lock	~~dl_dyn_init~~	
~~wn_sense_help~~			dl_item_dim	~~dl_key~~	
wn_visible			dl_set_item_flags		
			dl_set_prompt	*dl_changed*	
~~wn_set~~			dl_take_focus	*dl_focus*	
~~wn_sense~~			dl_handle_to_index	*dl_launch_sub*	
~~wn_draw~~			dl_index_to_handle	*dl_item_new*	
wn_init					

The ATSDIAL *class is not present on the Series 3.*

The ATSDIAL class implements ATS dialogs, that is, dialogs that are presented in response to an inter-process message from another, *controlling*, process. The ATS dialog mechanism is intended to be used only via the mechanisms supplied within the HWIM library (see, for example, the description of the `ws_do_remote_dial` method in the *WSERV Class* chapter. *An application should not subclass* ATSDIAL.

An ATS dialog may be dependent on two items of data that are copied from the controlling process. The first of these must be the resource for the dialog itself (considered to be an HD_DLGBOX_RSC struct). The second may be one of:

- a choice list resource
- an action list resource
- editable text for an edit box.

The ATS dialog may therefore contain, in addition to controls of other types, not more than one control selected from these three types. None of the other controls may be of any type that requires additional data (for example, from another resource file item).

See the *Series 3a Automatic Test System* chapter of the *Object Oriented Programming Guide* for a description of the use of ATS.

Class definition

Defined in sub-category file *xdlgbox.cl* (generated header file *xdlgbox.g*).

```
CLASS atsdial dlgbox
    {
    REPLACE destroy
    REPLACE dl_item_add
    REPLACE dl_dyn_init
    REPLACE dl_key

    PROPERTY
        {
        VOID *owner;
        VOID *main;
        VOID *buts;
        PR_WIN *filter;
        WORD filmethod;
        WORD chlist;
        WORD pid;
        WORD ret;
        UWORD locked;
        }
```

Property

`atsdial.owner`	If not NULL, the handle of an instance of the ATSSV class (see the *XADD Reference* manual).
`atsdial.main`	A pointer to an allocated memory cell containing the main resource data for the dialog.
`atsdial.buts`	If not NULL, a pointer to an allocated memory cell containing resource data for the dialog's one control that needs such data. The control may be a choice list, an action list or an edit box.
`atsdial.filter`	A preserved copy of the application's `w_ws->wserv.filter` property.
`atsdial.filmethod`	A preserved copy of the application's `w_ws->wserv.filmethod` property.
`atsdial.chlist`	Either zero or one plus the index of the control that makes use of the second item of data copied over from the controlling process. If positive, the control is a choice list, if negative the control is an edit box.
`atsdial.pid`	The process ID of the process that launched the ATS dialog.
`atsdial.ret`	A terminating value that can be sent in an SV_FREE message to `atsdial.owner`.
`atsdial.locked`	Normally TRUE, indicating that the ATS dialog code has incremented the application's DatLocked magic static.

ATSDIAL methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the ATS dialog and its resources. The action is as follows:

- Frees the allocated memory cell pointed to by `atsdial.main` and, if it exists, the allocated memory pointed to by `atsdial.buts`.
- Restores the values of `w_ws->wserv.filter` and `w_ws->wserv.filmethod` from the preserved values in `atsdial.filter` and `atsdial.filmethod`.
- If `atsdial.owner` is not `NULL`, sends it an `SV_FREE` message, passing the value of `atsdial.ret`.
- If `atsdial.locked` is non-zero, decrements `DatLocked`.
- Supersends the `DESTROY` message.

DL_ITEM_ADD — Add an item

```
VOID dl_item_add(AD_DLGBOX *p);
```

Mark the dialog as being started by the ATS mechanism by ORing the flag `WIN_FROM_ATS` into `win.flags` (for use by any `ACLIST` or `CHLIST` component) and then supersend the `DL_ITEM_ADD` message.

DL_DYN_INIT — Dynamic initialisation

```
INT dl_dyn_init(ATS_MESS *pm, VOID *owner);
```

Copies the value of `owner` into `atsdial.owner`, and also copies `w_ws->wserv.filter` and `w_ws->wserv.filmethod` into `atsdial.filter` and `atsdial.filmethod`.

If there is a keyboard filter that is non-permanent (`atsdial.filmethod` is not negative), the method then calls `p_leave(E_GEN_IN_USE)`. An application with a temporary keyboard filter is deemed not to be in a suitable state to handle an ATS dialog.

The process ID of the controlling process is copied from `pm->mess.pid` into `atsdial.pid`, and the application's filter is cleared (`w_ws->wserv.filter` and `w_ws->wserv.filmethod` are cleared). The value of `atsdial.ret` is set to -1, the default value that will indicate termination of the dialog by pressing Esc.

If `pm->u.d.butslen` is negative, this is taken to mean that `pm->u.buts` contains a pointer to initial text for an edit box. The value of `pm->u.d.butslen` is copied into `atsdial.chlist`, for later use, and `pm->u.butslen` is set to 2 (the length of a pointer).

Memory is allocated to contain the resources specified by `pm->u.d.main` (the main dialog resource) and `pm-u.d.buts` (optional data for one of the dialog's controls) The data is copied from the controlling process and pointers to the two allocated cells are written to `atsdial.main` and `atsdial.buts`.

The flags `DLGBOX_NO_WAIT` and `DLGBOX_NO_DDP` are ORed into the dialog box flags in the main dialog resource and the dialog box sends itself a `DL_INIT` message.

If `atsdial.chlist` is negative, the dialog's edit box is seeded with up to `ATS_MAX_EDITOR_LEN` bytes of text read over from the controlling process, at an offset that is specified by the two-byte cell pointed to by `atsdial.buts`.

The dialog then sends itself a `DL_SET_SIZE` message and calls `hInitVis`. The flag `PR_WIN_INITIALISED` is ORed into `win.flags`, the dialog sends `w_ws` a `WS_ADD_DIAL` message, `DatLocked` is incremented and `atsdial.locked` is set `TRUE`.

The method returns zero to indicate that `p_leave` has not been called. The method is thus suitable for being called under the protection of `p_enter`.

DL_KEY — Handle key input

```
INT dl_key(VOID);
```

Handle the Enter key input that exits the dialog.

If `atsdial.chlist` is positive (the dialog is presenting a choice list) write the index of the currently selected choice list item to `atsdial.ret` and return `WN_KEY_CHANGED`.

If `atsdial.chlist` is negative (the dialog is presenting an edit box) copy the text back to the controlling process, overwriting the string used to intialise the edit box, write zero to `atsdial.ret` and return `WN_KEY_CHANGED`.

If `atsdial.chlist` is zero, just return `WN_KEY_CHANGED`.

CHAPTER 8

LABELS, BUTTONS AND CHOICE LISTS

This chapter describes three of the basic components of a dialog box.

The TEXTWIN class, as its name suggests, is used to display static text, including any dialog box title and optional prompt messages for other controls. Dialog box buttons, used to exit the dialog box or to initiate other actions, are implemented by either the ACLIST or SMACLIST 'action list' classes. The CHLIST class provides the means of selecting one option from a number of alternatives.

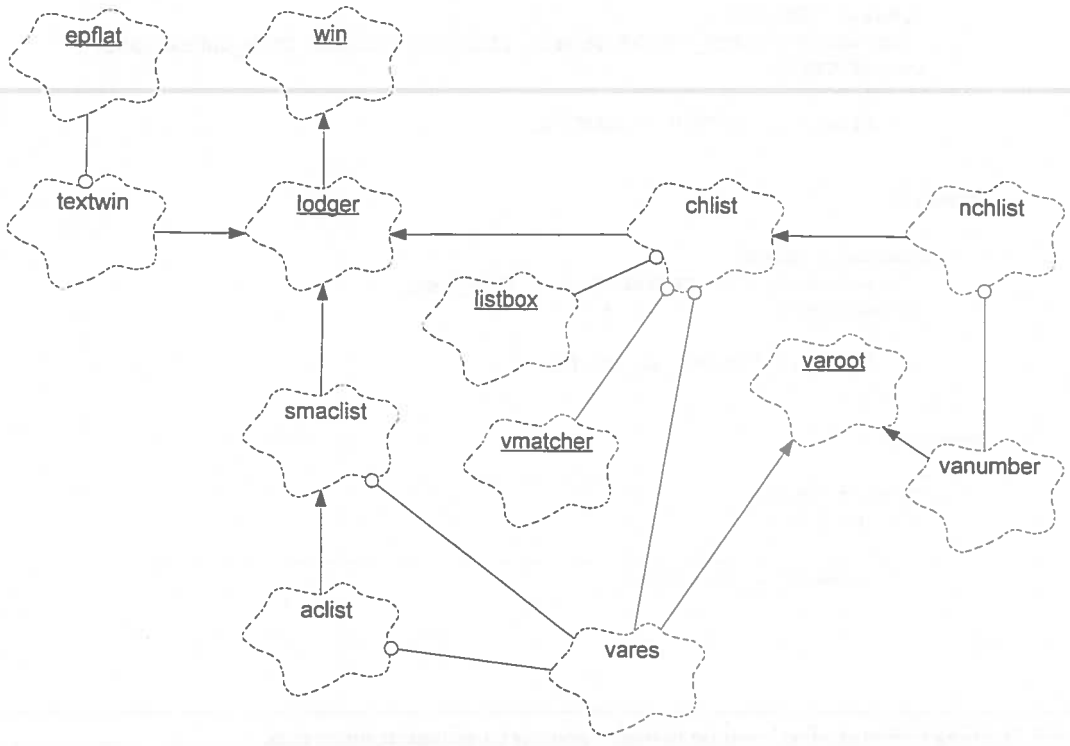
It is unlikely that an application will need to subclass any of the classes described in this chapter and it is expected that the vast majority of applications will simply use them as standard dialog components. In consequence applications will generally not send explicit messages to instances of these classes and it is therefore not necessary to understand the class methods in any great depth.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the WIN and LODGER classes, described in the *Windows* chapter
- the description of the DLGBOX class in the *Dialog Boxes* chapter, particularly the `wn_key` method which may sent WN_KEY messages to a dialog box component and respond to the return value
- for the CHLIST class, the description of its LISTBOX component in the *List Boxes and Menus* chapter

Class diagram



The TEXTWIN class

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy lg_draw lg_self_check lg_set_id_pos ~~wn_init~~ wn_visible ~~*lg_sense_width*~~ *lg_update*

TEXTWIN
label state
lg_sense_width wn_draw wn_emphasise wn_init wn_key wn_sense wn_set

An instance of TEXTWIN is used within a dialog box to display static text. In addition to displaying simple informational messages, it may be used as a prompt (or label) to a control, or as the dialog box title.

Text may be specified in a dialog resource in a number of ways. For example, the system error dialog, whose resource is listed below, simply declares two TEXTWIN components:

```
RESOURCE DIALOG sys_error_dialog
    {
    flags=DLGBOX_RBUF_FILLED|DLGBOX_NO_DDP;
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD|DLGBOX_ITEM_UNDERLINED;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_ac_continue;
                };
            }
        };
    }
```

where the TXTMESS resource struct and its default values are defined in *hwim.h* as:

```
STRUCT TXTMESS
    {
    WORD flags=0;
    TEXT str="";
    }
```

The first of the TEXTWIN components sets the flag DLGBOX_ITEM_UNDERLINED so that it appears as a dialog box title. The text of both components is determined by the type of the run-time error that causes the dialog to be displayed and is supplied dynamically.

A TEXTWIN dialog title may be specified more simply by providing text for the title element, as in the following example. This example also shows how to specify a TEXTWIN prompt to a control, by providing text for the prompt item of a CONTROL resource. In both of these cases the TEXTWIN flag values are set as appropriate by system code.

```
RESOURCE DIALOG sys_printer_model
    {
    title="Set printer";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            prompt="Select printer";
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            class=C_CHLIST;
            info=CHLIST{};
            },
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Default font";
            info=TXTMESS
                {
                flags=IN_TEXTWIN_POPOUT;
                };
            }
        };
    }
```

Class definition

Defined in sub-category file *textwin.cl* (generated header file *textwin.g*).

```
CLASS   textwin         lodger
Text window
    {
    REPLACE wn_draw
    REPLACE wn_init
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_key
    REPLACE lg_sense_width
    REPLACE wn_emphasise

    CONSTANTS
        {
        PR_TEXTWIN_AL_LEFT              0x00
        PR_TEXTWIN_AL_RIGHT             0x01
        PR_TEXTWIN_AL_CENTRE            0x02
        PR_TEXTWIN_BOLD                 0x04
        PR_TEXTWIN_BULLET               0x20
        PR_TEXTWIN_POPOUT               0x40
        PR_TEXTWIN_FLASHING             0x80

        IN_TEXTWIN_AL_LEFT              PR_TEXTWIN_AL_LEFT
        IN_TEXTWIN_AL_RIGHT             PR_TEXTWIN_AL_RIGHT
        IN_TEXTWIN_AL_CENTRE            PR_TEXTWIN_AL_CENTRE
        IN_TEXTWIN_BOLD                 PR_TEXTWIN_BOLD
        IN_TEXTWIN_BULLET               PR_TEXTWIN_BULLET
        IN_TEXTWIN_POPOUT               PR_TEXTWIN_POPOUT
```

```
        SE_TEXTWIN_ALIGN            (PR_TEXTWIN_AL_RIGHT|PR_TEXTWIN_AL_CENTRE)
        SE_TEXTWIN_BOLD             PR_TEXTWIN_BOLD
        SE_TEXTWIN_TEXT             0x08
        SE_TEXTWIN_BULLET           PR_TEXTWIN_BULLET
        }

    TYPES
        {
        typedef struct
            {
            UWORD state;
            TEXT label[1];
            }IN_TEXTWIN;
        typedef struct
            {
            INT flags;
            UWORD state; Alignment, underline
            TEXT *buf;
            UWORD len;
            }SE_TEXTWIN;
        }

    PROPERTY 1
        {
        PR_EPFLAT *label;       label text object
        UWORD state;            Alignment, underline
        }
    }
```

The flag PR_TEXTWIN_FLASHING is not defined on the Series 3.

Property

textwin.label	either NULL or the handle of the instance of EPFLAT that contains the text
textwin.state	any sensible combination of the flags listed below

TEXTWIN flags

PR_TEXTWIN_AL_LEFT	if set, the text is aligned left. No other PR_TEXTWIN_AL_XXX flag should be set
PR_TEXTWIN_AL_RIGHT	if set, the text is aligned right. No other PR_TEXTWIN_AL_XXX flag should be set
PR_TEXTWIN_AL_CENTRE	if set, the text is centred. No other PR_TEXTWIN_AL_XXX flag should be set
PR_TEXTWIN_BOLD	if set, the text is to be drawn in a bold font
PR_TEXTWIN_BULLET	if set, the text is to be preceded by a rectangular bullet
PR_TEXTWIN_POPOUT	if set, the text window is to trigger a 'pop-out' subdialog on receiving a Tab keypress
PR_TEXTWIN_FLASHING	If set, use a flashing text cursor, otherwise any cursor is a highlighted obloid. This flag is not available on the Series 3, which is restricted to a non-flashing cursor.

TEXTWIN methods

WN_INIT — Initialise

```
VOID wn_init(IN_TEXTWIN *par, PR_WIN *landlord);
```

Initialise the text window to contain any text specified by the IN_TEXTWIN struct pointed to by par.

Sets lodger.landlord to the value of landlord and textwin.state to par->state.

If the par->label buffer does not contain a null string, an instance of EPFLAT is created and initialised to contain the specified text, as indicated by the following code:

```
p_send3(epflat,O_EP_INIT,WS_MAX_PRINT_BOX_TEXT_LEN);
p_send4(epflat,O_EP_SET_TEXT,&par->label[0],p_slen(&par->label[0]));
```

If this is successful, the handle of the instance of EPFLAT is stored in textwin.label. On failure p_leave is called.

This message is sent from the DLGBOX dl_item_add method.

WN_DRAW — Draw text

```
VOID wn_draw(PR_TEXTWIN *self);
```

Draw the text of the text window, assuming the existence of an appropriate graphics context.

If textwin.state contains PR_TEXTWIN_BOLD, the graphics context is set to use a bold font.

On the Work*about*, if the control is being used as a dialog component (win.flags contains PR_WIN_IS_DLCTRL) and the dialog is set to use the small font (the owning dialog box, whose handle is stored in lodger.landlord, has a non-zero value of dlgbox.font) the graphics context is set to use the small font.

The text is sensed by sending textwin.label an EF_SENSE_BUF message and is then drawn, with a call to gPrintBoxText, at a position within the text window determined by the PR_TEXTWIN_AL_XXX flag in textwin.state. The bullet rectangle is drawn or cleared, depending on the presence or absence of the PR_TEXTWIN_BULLET flag in textwin.state. If win.flags contains PR_WIN_EMPHASISED, the text (and any bullet) is highlighted. If textwin.state contains PR_TEXTWIN_FLASHING, the highlight takes the form of a flashing cursor.

If the first character in the string is 0x01 then the second character is interpreted as an offset in pixels for the remaining characters. This technique is used to align the pack selector prompt as illustrated in the following picture:



The "Disk" text has been offset so as to align it with the "Name" text.

If textwin.state contains PR_TEXTWIN_BOLD, the graphics context is reset to use a normal font.

This message is sent from the DLGBOX wn_draw method.

WN_SET — Set property

```
VOID wn_set(SE_TEXTWIN *txtset);
```

Set the text and/or the textwin.state flags according to the data in the SE_TEXTWIN struct pointed to by txtset. The items that are to be modified are specified by txtset->flags, which should be a combination of SE_TEXTWIN_XXX flags. The value of txtset->state should be a combination of PR_TEXTWIN_XXX flags.

If txtset->flags contains SE_TEXTWIN_TEXT, the method sets the text (by sending textwin.label an EP_SET_TEXT message) to the first txtset->len bytes in the buffer pointed to by txtset->buf. If textwin.label is NULL, an instance of EPFLAT is created and initialised (as described in the wn_init method) before the EP_SET_TEXT message is sent.

Each of the remaining flags in txtset->flags causes the corresponding PR_TEXTWIN_XXX bit in txtset->state to be written (either set or cleared) to textwin.state.

An LG_DRAW message is sent, causing the text window to be redrawn, if txtset->flags contains any of the flags SE_TEXTWIN_TEXT, SE_TEXTWIN_ALIGN, SE_TEXTWIN_BOLD or SE_TEXTWIN_BULLET.

The following example sets the text to the string pointed to by str. It also clears any PR_TEXTWIN_BOLD flag and sets PR_TEXTWIN_AL_RIGHT (clearing PR_TEXTWIN_AL_LEFT and PR_TEXTWIN_AL_CENTRE in the process). No other flags in textwin.state are affected.

```
LOCAL_C VOID SetTextWin(PR_TEXTWIN *textwin,TEXT *str)
    {
    SE_TEXTWIN set;

    set.flags=SE_TEXTWIN_TEXT|SE_TEXTWIN_ALIGN|SE_TEXTWIN_BOLD;
    set.state=PR_TEXTWIN_AL_RIGHT; /* PR_TEXTWIN_BOLD not set, so will be cleared */
    set.buf=str;
    set.len=p_slen(str);
    p_send3(textwin,O_WN_SET,&set);
    }
```

It is rare for application code to set more than the text of an instance of TEXTWIN used as a dialog box component. The flags in textwin.state are generally either set on initialisation (usually from IN_TEXTWIN_XXX values set in a resource item) or are set or cleared by system code (see, for example, the DLGBOX dl_item_lock and dl_item_dim methods).

This message is sent from the DLGBOX wn_set method, normally invoked from application code by means of the hDlgSet and hDlgSetText utility functions.

WN_SENSE — Sense text

```
VOID wn_sense(SE_TEXTWIN *txtset);
```

Write a pointer to the text in txtset->buf and the length of the text to txtset->len, according to the code:

```
txtset->len=p_send3(self->textwin.label,O_EF_SENSE_BUF,&txtset->buf);
```

On the Series 3, this method must not be called if the text window contains no text. On other machines the method simply writes zero to txtset->len if the text window contains no text.

This message is sent from the DLGBOX wn_sense method, normally invoked from application code by means of the hDlgSense utility function.

WN_KEY — Handle key input

```
INT wn_key(INT keycode, INT modifiers);
```

Simply returns WN_KEY_NO_CHANGE (0) unless textwin.state contains PR_TEXTWIN_POPOUT.

If textwin.state contains PR_TEXTWIN_POPOUT, the text window can trigger a subdialog (see the discussion of subdialogs in the *Dialog Boxes* chapter) on receipt of a WN_KEY message with keycode set to W_KEY_TAB. It does this by writing to wserv.subdial a value of one plus the text window's index within the dialog (the index is found by sending the dialog box a DL_HANDLE_TO_INDEX message). For any other value of keycode, the method uses the hInfoPrint utility function to display the information message with system resource ID SYS_POPOUT_HELP (in English, this is the message "Press Tab to change this item").

In all cases the method returns WN_KEY_NO_CHANGE (0).

This message is sent from the DLGBOX wn_key method.

LG_SENSE_WIDTH — Return required control width

```
INT lg_sense_width(VOID);
```

If `textwin.label` is `NULL`, returns zero.

Otherwise, senses the text window's text by sending `textwin.label` an `EF_SENSE_BUF` message and returns the width required to draw the string, plus an allowance for a preceding bullet (regardless of whether a bullet is actually present).

This message is sent from the `DLGBOX` `dl_set_size` method.

WN_EMPHASISE — Emphasise

```
VOID wn_emphasise(UINT flag);
```

Emphasise the window if `flag` is `TRUE`, or remove the emphasis if `flag` is `FALSE`. Does nothing if attempting to set the emphasis to its current state, or if `textwin.label` is `NULL`.

Sets the `PR_WIN_EMPHASISED` bit in `win.flags` if `flag` is `TRUE`, otherwise clears this bit and erases any text cursor.

This message is sent from the `DLGBOX` `dl_take_focus` method.

The SMACLIST class

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw wn_emphasise ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ *wn_set* ~~*wn_sense*~~ ~~*wn_draw*~~ ~~*wn_init*~~	~~destroy~~ lg_draw lg_self_check ~~lg_set_id_pos~~ ~~wn_init~~ wn_visible ~~*lg_sense_width*~~ *lg_update*

SMACLIST
data num pos totalwidth
destroy lg_sense_width lg_set_id_pos wn_draw wn_init wn_key wn_sense

`SMACLIST` is the 'small' form of an action list. It is used to implement a set of buttons in cases where there is a requirement for the dialog to occupy as small a region of the screen as possible. In consequence, a dialog containing an instance of `SMACLIST` usually does not contain any other controls.

The following diagram shows an example of the use of `SMACLIST` in the Replace command of the Word application, to control how text replacements are made. In this case the dialog must be small since there is a requirement to display as much as possible of the text in which replacements are being made.



The dialog requires the two resources given below.

```
RESOURCE ACLIST_ARRAY repl_ac
    {
    button =
        {
        PUSH_BUT
            {
            keycode=-'e';
            str="End";
            },
        PUSH_BUT
            {
            keycode='r';
            str="Replace";
            },
        PUSH_BUT
            {
            keycode='s';
            str="Skip";
            },
        PUSH_BUT
            {
            keycode='a';
            str="All";
            }
        };
    }
```

where the ACLIST_ARRAY and PUSH_BUT resource structs are defined in *hwim.rh* as:

```
STRUCT ACLIST_ARRAY
    {
    LEN BYTE STRUCT button[];   /* array of push_buttons */
    }

STRUCT PUSH_BUT
    BYTE {
    WORD keycode;
    TEXT str;       /* text associated with button */
    }
```

An ACLIST_ARRAY resource defines, for each button, the key code that activates the button and the button's associated text. It must contain at least one PUSH_BUT struct.

The keycode value may be supplied in either upper or lower case, or may be a W_KEY_XXX value. If no keycode is set to W_KEY_ESCAPE, a negative keycode value, such as in the first item in the above example, indicates that the button may also be activated by pressing Esc. Only one such item may be included in the array.

The following example illustrates the corresponding DIALOG resource. If a dialog contains other controls, the SMACLIST control must appear last.

```
RESOURCE DIALOG repl_inst_dl
    {
    flags=PR_WIN_FORCE_BOTTOM|PR_WIN_FORCE_LEFT;
    controls=
        {
        CONTROL
            {
            class=C_SMACLIST;
            info=ACLIST
                {
                rid=repl_ac;
                };
            }
        };
    }
```

where the ACLIST resource struct is defined in *hwim.rh* as:

```
STRUCT ACLIST  /* Initialising struct for action list */
    {
    LINK rid;
    }
```

Class definition

Defined in sub-category file *aclist.cl* (generated header file *aclist.g*).

```
CLASS   smaclist     lodger
small action list lodges within dialog. No real button will be drawn
    {
    REPLACE destroy
    REPLACE wn_init           load action list, set dead & centre flags
    REPLACE wn_key            process key press
    REPLACE wn_draw           print text of current selection
    REPLACE wn_sense=p_dummy
    REPLACE lg_set_id_pos     calculate position of button
    REPLACE lg_sense_width    ret min width of control, real width may alter again
later

    CONSTANTS
        {
        SMACLIST_MAX_STRING 80  /* max of 80 characters */
        }
    TYPES
        {
        typedef struct
            {
            UWORD width; width of ACLIST_button or letter within bracket for SMACLIST
            UWORD x;     tl x-offset of ACLIST_but or '(' in case of SMACLIST
            } BUT_POS;
        typedef struct
            {
            WORD rid;   resource ID of lbc text-array
            } IN_ACLIST;
        }

    PROPERTY 1
        {
        PR_VARES *data;   array holding keycode & text outside button
        BUT_POS *pos;     array holding each button's width & x-offset
        UWORD totalwidth; cumulative total of button widths, excluding all gaps
        UWORD num;        number of items in action list
        UWORD lwidth;     width of left bracket
        }
    }
```

The Series 3 version defines the BUT_POS struct as:

```
typedef struct
    {
    UWORD width;
    UWORD x;
    } BUT_POS;
```

The Series 3 version also contains the following additional defined constants:

```
SMACLIST_MIN_GAP    12  /* 2 numeric spaces between adjacent text items */
LEFT_BRACKET_WIDTH  4
```

On the Series 3a the corresponding values are read or derived from the globally accessible data that is described in the *Introduction* chapter of this manual. The left bracket width is stored in `smaclist.lwidth` - this property does not exist in the Series 3 SMACLIST class definition.

Property

`smaclist.data`	the handle of an instance of a VARES array, containing the key code and associated text for each button

smaclist.pos	a pointer to an allocated heap cell containing an array of the width and offset of each button. In the SMACLIST class, smaclist.pos->x is the x-offset of the button's leading left bracket character and smaclist.pos->width is the width of the single character that represents the button's key code
smaclist.totalwidth	the basic width of the control's buttons, including the smallest allowed button spacing. The actual width of the control, contained in lodger.width, may be greater than smaclist.totalwidth
smaclist.num	a count of the number of buttons in the action list
smaclist.lwidth	the width of a left bracket

SMACLIST methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Free the heap cell pointed to by smaclist.pos and supersend the DESTROY message.

This message is sent from the DLGBOX destroy method.

WN_INIT — Initialise

```
VOID wn_init(IN_SMACLIST *init, PR_DLGBOX *landlord);
```

Sets lodger.landlord to the passed value of landlord, which is assumed to be the handle of an instance of (a subclass of) DLGBOX and ORS DLGBOX_SMALL_ACTION_LIST into landlord->dlgbox.flags.

Creates an instance of VARES, then (provided init->rid is not zero) loads the resource specified by the resource ID init->rid and uses it to initialise the instance of VARES (by means of a VA_INIT message). If this succeeds the handle of the instance of VARES is written to smaclist.data. The resource, when loaded, is regarded as a sequence of PUSH_BUT structs, this struct being defined in the ACLIST class definition as:

```
typedef struct
    {
    WORD keycode; /* keycode to match text of inside button */
    TEXT str[1];  /* text above the button */
    } PUSH_BUT;   /* correspond to push_but in hwim.rh */
```

It corresponds to the PUSH_BUT resource struct used to construct an ACLIST_ARRAY resource (both of these resource structs are defined in *hwim.rh*).

On the Series 3a, if landlord->win.flags contains WIN_FROM_ATS (which implies that the landlord is an instance of ATSDIAL or a subclass) the data in the VARES instance pointed to by smaclist.data is loaded with button data from the ATSDIAL instance.

The number of buttons is found by sending smaclist.data a VA_COUNT message and is written to smaclist.num. A heap cell is allocated to contain the width an offset of each button and its address is written to smaclist.pos.

The dialog box flags for this item are set to contain DLGBOX_ITEM_CENTRE, DLGBOX_ITEM_DEAD and (but not on the Series 3) DLGBOX_ITEM_ACLIST by sending lodger.landlord a DL_SET_ITEM_FLAGS message.

The method calls p_leave on failure to create or initialise the smaclist.data component or to allocate the smaclist.pos heap cell.

This message is sent from the DLGBOX dl_item_add method.

WN_KEY — Handle key input

```
INT wn_key(INT keycode, INT modifiers);
```

Scan the items in the `smaclist.data` array for a match between the stored key code and the passed `keycode` value. If the stored key code is negative it is considered to match with either the correponding (positive) value of keycode or the keycode value `W_KEY_ESCAPE`.

If there is a match, the corresponding button is animated to give a visual response to the keypress and the method then returns the index (the leftmost button has an index of zero) of the matching button.

If there is no match the method returns -1.

This method is not subclassed by `ACLIST`. It supplies the button animation for both the `SMACLIST` and `ACLIST` classes, distinguishing the two cases by testing `lodger.landlord->dlgbox.flags` for the presence of `DLGBOX_ACTION_LIST`. In the `SMACLIST` case the button's key code character is animated by highlighting the button text for a period of two tenths of a second. For `ACLIST` the animation consists of a call to `wDrawButton` to draw the button in a depressed state, followed two tenths of a second later by another call to redraw the button in its undepressed state. In all cases a temporary graphics context is created for the animation and freed again afterwards.

This message is sent from the `DLGBOX` `wn_key` method.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the row of buttons, assuming the existence of an appropriate graphics context.

The method constructs a text string containing the buttons and displays it with a call to `gPrintBoxText`. The data for each button is read from the `smaclist.data` array. The text of each button consists of the bracketed character corresponding to the button's key code, followed by the button text. Consecutive buttons are separated by two numeric spaces.

This message is sent from the `DLGBOX` `wn_draw` method.

WN_SENSE — Sense data

```
VOID wn_sense(SE_CHLIST *se);
```

An instance of `SMACLIST` has no data that can usefully be sensed, so the supplied method does nothing.

This method is supplied since it is called by system code.

LG_SET_ID_POS — Set id, width and button positions

```
VOID lg_set_id_pos(INT id, P_POINT *pos, UINT width);
```

Supersend the `LG_SET_ID_POS` message and then, provided that `smaclist.totalwidth` (the width occupied by the buttons themselves) is less than `lodger.width`, adjust the button positions to centre them in the lodger window.

This method is not subclassed by `ACLIST`. It supplies the button positioning logic for both the `SMACLIST` and `ACLIST` classes, distinguishing the two cases by testing `lodger.landlord->dlgbox.flags` for the presence of `DLGBOX_ACTION_LIST`. In the `SMACLIST` case the button separation is kept fixed but the buttons are centred by adjusting the value of `lodger.offset.x`. For `ACLIST` the button spacing is increased (by adjusting the x position of each button in the `smaclist.pos` array) as well as centering the buttons by changing `lodger.offset.x`.

This message is sent from the `DLGBOX` `dl_set_size` method.

LG_SENSE_WIDTH — Sense required width

```
INT lg_sense_width(VOID);
```

Calculate the minimum width required to draw the contents.

The content of the `smaclist.pos` element for each button is set so that `smaclist.pos->x` contains the x-offset to the leading left bracket of the button and `smaclist.pos->width` contains the pixel width of the uppercased character representing the button's key code.

The display width of each button is calculated as the width of the uppercased key code character, including its enclosing brackets, plus the width of the following string. The button widths are accumulated into `smaclist.totalwidth`, together with a fixed separation between the buttons. The method assumes that `smaclist.totalwidth` initially contains zero, that is, it assumes that the method is called only once in the lifetime of an instance.

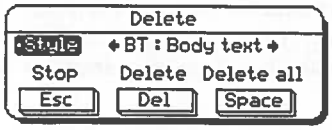
The method returns the width of the control (which is also the final value stored in `smaclist.totalwidth`).

This message is sent from the DLGBOX `dl_set_size` method.

The ACLIST class

WIN	LODGER	SMACLIST		ACLIST
flags id	landlord offset width	data num pos totalwidth		text
~~destroy~~ wn_calc_position wn_connect wn_dodraw wn_emphasise ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ *wn_set* ~~*wn_sense*~~ ~~*wn_draw*~~ ~~*wn_init*~~	~~destroy~~ lg_draw lg_self_check ~~lg_set_id_pos~~ ~~wn_init~~ wn_visible ~~*lg_sense_width*~~ *lg_update*	destroy ~~lg_sense_width~~ lg_set_id_pos ~~wn_draw~~ ~~wn_init~~ wn_key wn_sense	1	lg_sense_width wn_draw wn_init

The ACLIST class is used to add an action list of one or more buttons to a dialog, in cases where screen space is not at a premium. Each button has two associated items of text; one drawn inside the button, representing the key press that activates it, and one drawn above the button, intended to inform the user of the action that the button initiates.



As for SMACLIST, an ACLIST control requires the buttons to be described in an ACLIST_ARRAY resource:

```
RESOURCE ACLIST_ARRAY delete_ac
    {
    button =
        {
        PUSH_BUT
            {
            keycode=W_KEY_ESCAPE;
            str="Stop";
            },
        PUSH_BUT
            {
            keycode=W_KEY_DELETE_LEFT;
            str="Delete";
            },
        PUSH_BUT
            {
            keycode=' ';
            str="Delete all";
            }
        };
    }
```

Again, the keycode of one element may be negative to allow it to be activated by pressing Esc as well as the specified key. The above example does not use this option since it explicitly uses a keycode of W_KEY_ESCAPE for one of its buttons.

The dialog resource for the dialog illustrated above is as follows:

```
RESOURCE DIALOG delete_dl
    {
    title="Delete";
    flags=DLGBOX_NOTIFY_ESCAPE|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            prompt="Style";
            info=CHLIST{}; /* choice list content supplied dynamically */
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=delete_ac;
                };
             }
        };
    }
```

As with SMACLIST, an ACLIST control must be the last component of the dialog.

Class definition

Defined in sub-category file *aclist.cl* (generated header file *aclist.g*).

```
CLASS   aclist   smaclist
action list lodges within dialog.It is equivalent to a list of push_buttons
    {
    REPLACE wn_init           load action list given rid & set dead & centre
    REPLACE wn_draw
    REPLACE lg_sense_width    returns min width of control

    TYPES
        {
        typedef struct
            {
            WORD keycode; keycode to match text of inside button
            TEXT str[1];  text above the button
            } PUSH_BUT;   corresponds to push_but in hwim.rh
        }

    PROPERTY 1
        {
        PR_VARES *text;   array holding the keycode & text for above each button
        WORD width        width of each button
        UWORD y1;         offset to top of upper text boxes
        UWORD y2;         offset to top of buttons themselves
        }
    }
```

The Series 3 version contains the following defined constants:

```
ACLIST_BUTTON_MIN_GAP          5  min gap between 2 buttons
ACLIST_BUTTON_WIDTH           38  For French 'Entrer'
ACLIST_BUTTON_HEIGHT          12
ACLIST_BUTTON_SHADOW           2
ACLIST_TOP_GUTTER              4  from self->lodger.offset.y
```

On the Series 3a the corresponding values are read or derived from the globally accessible data that is described in the *Introduction* chapter of this manual.

The property items:

```
WORD width        width of each button
UWORD y1;         offset to top of upper text boxes
UWORD y2;         offset to top of buttons themselves
```

do not exist in the Series 3 ACLIST class definition.

Property

`aclist.text` — The handle of an instance of a VARES array, containing the key code and associated text that is to be displayed above each button.

`aclist.width` — Used internally on the Series 3a to store the (fixed) width of all buttons. On the Series 3 this width is represented by the constant value ACLIST_BUTTON_WIDTH.

`aclist.y1` — Used internally on the Series 3a to store the (fixed) y-offset within the control's window to the top of the rectangle containing the text that is displayed above a button. On the Series 3 this is calculated from the sum of `lodger.offset.y` and ACLIST_TOP_GUTTER.

`aclist.y2` — Used internally on the Series 3a to store the (fixed) y-offset within the control's window to the top of the rectangle containing a button. On the Series 3 this is calculated from the sum of `lodger.offset.y`, ACLIST_TOP_GUTTER and ACLIST_BUTTON_HEIGHT.

The superclass property `smaclist.data` is set to the handle of an instance of VARES that contains an array of key codes and text (to appear inside the button) for standard buttons. These are loaded from the `SYS_BUTTON_TEXT` system resource, which contains the following standard buttons:

Keycode	Button text (English version)
`W_KEY_RETURN`	Enter
`W_KEY_ESCAPE`	Esc
`W_KEY_DELETE_LEFT`	Del
`W_KEY_SPACE`	Space
`W_KEY_UP`	↑
`W_KEY_DOWN`	↓
`W_KEY_RIGHT`	→
`W_KEY_LEFT`	←
`W_KEY_TAB`	Tab
`W_KEY_MENU`	Menu

ACLIST methods

WN_INIT — Initialise

```
VOID wn_init(IN_ACLIST *init, PR_DLGBOX *landlord);
```

Sets `DLGBOX_ACTION_LIST` in `landlord->dlgbox.flags`, creates an instance of VARES and loads into it the standard button data from the `SYS_BUTTON_TEXT` system resource. If this is successful, the VARES handle is written to `aclist.text`.

On the Work*about* `lodger.landlord` is set to the passed value of `landlord` (as is done on all machines in the `SMACLIST` `wn_init` method) so that the control can determine if it should draw itself using the small font.

Finally, the method supersends the `WN_INIT` message and, on the Series 3a, clears the flag `DLGBOX_SMALL_ACTION_LIST` that is set in `landlord->dlgbox.flags` by the superclass `wn_init` method. Note that this means that a Series 3 `ACLIST` sets both the `DLGBOX_ACTION_LIST` and `DLGBOX_SMALL_ACTION_LIST` flags in `landlord->dlgbox.flags` (system code that must distinguish between the two classes always tests for the presence of `DLGBOX_ACTION_LIST`).

This message is sent from the `DLGBOX` `dl_item_add` method.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the row of buttons, assuming the existence of an appropriate graphics context.

For each button, the method uses `gPrintBoxText` to draw centred text above the button, the text (and the key code) being read - via a `VA_PBUF` message - from the corresponding element of the VARES array whose handle is stored in `smaclist.data`. Each button is drawn with a call to `wDrawButton`, the text for the button being read from the element of the `aclist.text` array whose key code matches the one read earlier from the button's `smaclist.data` array element. If the button's key code does not match with any of the standard buttons, its text is taken to be the uppercased key code.

This message is sent from the `DLGBOX` `wn_draw` method.

LG_SENSE_WIDTH — Sense required width

```
INT lg_sense_width(VOID);
```

Calculate the minimum width required to draw the contents.

The method calculates the width of each button, sufficient to contain its text, subject to each button having a minimum width that is dependent on the machine type. These widths are stored in successive elements of the array pointed to by `smaclist.pos`. The button separations are calculated so that they are evenly spaced, with a minimum separation that is dependent on the machine type. The corresponding button positions are

also stored in the `smaclist.pos` array. Note that the button positioning logic pays no attention to the widths of the labels that appear above the buttons. It is the responsibility of the application programmer to select the text for these items so that they do not overlap.

The method writes the sum of the widths of the buttons and the inter-button gaps to `smaclist.totalwidth` and returns this value.

The width of the control may subsequently be increased, on receipt of an `LG_SET_ID_POS` message, with the control's actual width then being stored in `lodger.width`.

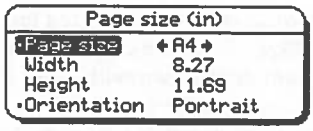
This message is sent from the `DLGBOX` `dl_sense_size` method.

The CHLIST class

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ lg_draw lg_self_check lg_set_id_pos ~~wn_init~~ wn_visible ~~*lg_sense_width*~~ *lg_update*

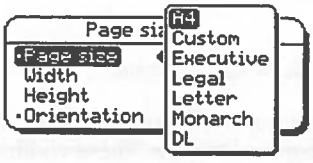
CHLIST
data flags matcher matchlen nsel pop
destroy lg_sense_width wn_draw wn_emphasise wn_init wn_key wn_sense wn_set

The `CHLIST` class provides a choice list control that can be used to select one of a range of options. A choice list appears as shown by the 'Page size' control in the following illustration, taken from the page size system dialog.



A particular option may be selected by means of the Home and End keys, the left and right cursor keys, or by first letter match. A choice list may additionally be configured to perform incremental matching with a sequence of key presses.

A choice list responds to the Tab key by displaying a pop-out expanded view, as shown below.



The simplest form of choice list has its items specified by `MENU` resource structs. The dialog illustrated above contains two choice list controls, with their resources defined in the system resource file as follows:

```
RESOURCE MENU sys_page_size
    {
    items=
        {
        CHOICE_ITEM {str="A4";},
        CHOICE_ITEM {str="Custom";},
        CHOICE_ITEM {str="Executive";},
        CHOICE_ITEM {str="Legal";},
        CHOICE_ITEM {str="Letter";},
        CHOICE_ITEM {str="Monarch";},
        CHOICE_ITEM {str="DL";}
        };
    }

RESOURCE MENU sys_orient
    {
    items=
        {
        CHOICE_ITEM {str="Portrait";},
        CHOICE_ITEM {str="Landscape";}
        };
    }
```

where the CHOICE_ITEM resource struct is defined in *hwim.rh* as:

```
STRUCT CHOICE_ITEM /* choice list item */
    BYTE {
    TEXT str=""; /* identification text */
    }
```

The corresponding system DIALOG resource is as follows where, for clarity, only the choice list control elements are shown:

```
RESOURCE DIALOG sys_pagesize_dl
    {
    title="Page size";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED|DLGBOX_APPEND_UNITS_TITLE;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            prompt="Page size";
            info=CHLIST{rid=sys_page_size;};
            },
        ...
        CONTROL
            {
            class=C_CHLIST;
            prompt="Orientation";
            info=CHLIST{rid=sys_orient;};
            }
        };
    }
```

where the CHLIST resource struct and its default values are defined in *hwim.h* as:

```
STRUCT CHLIST
    {
    LINK rid=0;
    BYTE nsel=0;
    BYTE flags=0;
    }
```

The content of a choice list may be set dynamically, using the wn_set method. This will typically be performed in the dialog's dl_dyn_init method. Changing the content of a choice list after the dialog box has become visible is not recommended, particularly if there is any risk that the new content may need a wider display.

Class definition

Defined in sub-category file *chlist.cl* (generated header file *chlist.g*).

```
CLASS   chlist     lodger
choice list
self->win.offset.x is tl.x of text (excluding left arrow)
    {
    REPLACE destroy
    REPLACE wn_key              process key press
    REPLACE wn_draw             print text of current selection
    REPLACE wn_set              set data,nsel or retain property
    REPLACE wn_sense            sense nsel value
    REPLACE wn_emphasise        draws arrow if highlighted
    REPLACE wn_init             create menu given rid
    REPLACE lg_sense_width      returns (min text width + right arrow).

    CONSTANTS
        {
        SE_CHLIST_NSEL          0x01
        SE_CHLIST_DATA          0x02
        SE_CHLIST_RETAIN        0x04
        IN_CHLIST_INCREMENTAL   0x01
        PR_CHLIST_RETAIN        0x04
        PR_CHLIST_SUSPENDED     0x08    To suspend display of cursor
        PR_CHLIST_FIXED_WIDTH   0x10
        }

    TYPES
        {
        typedef struct
            {
            WORD rid;           resource ID of lbc text-array
            UBYTE nsel;         initial value of nsel
            UBYTE flags;        IN_CHLIST_INCREMENTAL
            } IN_CHLIST;

        typedef struct
            {
            UWORD set_flags;    which fields significant
            PR_VAROOT *data;    possible new data value
            UWORD nsel;         possible new nsel value
            } SE_CHLIST;
        }

    PROPERTY 2
        {
        PR_LISTBOX *pop;        handle of pop-out menu (NULL if un-squirted mode)
        PR_VMATCHER *matcher;   set if incremental matching required
        PR_VARES *data;         pointer to lbc text-array struct
        UWORD nsel;             selected item in list (zero for first)
        UWORD flags;            holds PR_CHLIST_ flags
        UWORD matchlen;         length of match string
        }
    }
```

The Series 3 version does not define PR_CHLIST_FIXED_WIDTH.

Property

chlist.pop	either NULL or the handle of an instance of LISTBOX, used to display a pop-out expanded view
chlist.matcher	either NULL or the handle of an instance of VMATCHER, used to perform incremental matching
chlist.data	the handle of an instance of VARES, containing the text for the choice list items
chlist.nsel	the index number of the currently selected item
chlist.flags	a combination of the PR_CHLIST_XXX flags described below

chlist.matchlen	if chlist.matcher is not NULL, the character offset to the incremental matching cursor, otherwise not used

CHLIST flags

PR_CHLIST_RETAIN	if set, the instance of VARES whose handle is stored in chlist.data will not be destroyed when replaced by another instance of VARES or when the choice list is destroyed
PR_CHLIST_SUSPENDED	when set, disables the drawing of any incremental matching cursor

CHLIST methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

If chlist.flags does not contain PR_CHLIST_RETAIN and chlist.data is not NULL, send chlist.data a DESTROY message. Then supersend the DESTROY.

This message is sent from the DLGBOX destroy method.

WN_INIT — Initialise

```
VOID wn_init(IN_CHLIST *init, PR_WIN *landlord);
```

Set lodger.landlord to the passed value of landlord. Create an instance of VARES and (provided init->rid is not zero) load into it the resource with ID init->rid. If the creation and initialisation is successful, write the handle of the instance to chlist.data.

On the Series 3a, if landlord->win.flags contains WIN_FROM_ATS (which implies that the landlord is an instance of ATSDIAL or a subclass) the data in the VARES instance pointed to by smaclist.data is loaded with data from the ATSDIAL instance, rather than from the resource.

If init->flags contains IN_CHLIST_INCREMENTAL, create and initialise an instance of VMATCHER, according to the following code:

```
p_send5(self->chlist.matcher,O_IM_INIT,&chlist.matchlen,80,chlist.data);
```

and store its handle in chlist.matcher.

If init->nsel is not zero, sets chlist.nsel to init->nsel, provided this is within the range of choice list items, otherwise sets to the last item. If this changes chlist.nsel and chlist.matcher is not NULL, sends chlist.matcher an IM_SET_VAL message, passing chlist.nsel, and sets chlist.matchlen to zero.

This message is sent from the DLGBOX dl_item_add method.

WN_KEY — Handle key input

```
INT wn_key (INT keycode, INT modifiers);
```

If the choice list contains no items, the method simply returns WN_KEY_NO_CHANGE (0).

If a pop-out expanded view is currently displayed, chlist.pop is sent the WN_KEY message. Subsequent processing depends on the return value from this message as follows:

WN_KEY_NO_CHANGE	the method simply returns WN_KEY_NO_CHANGE
any positive value	this indicates the selection of a new item in the choice list, and chlist.nsel is set appropriately. The method then returns WN_KEY_CHANGED
any other (-ve) value	chlist.pop is sent a DESTROY message, chlist.pop is set to NULL, and the method returns the message return value

Otherwise, processing is dependent on the value of keycode, as follows:

W_KEY_LEFT	chlist.nsel is decremented by one or, if already zero, set to the last item in the choice list. The method sends an LG_DRAW message and returns WN_KEY_CHANGED
W_KEY_RIGHT	chlist.nsel is incremented by one or, if already at the last item in the choice list, set to zero. The method sends an LG_DRAW message and returns WN_KEY_CHANGED
W_KEY_HOME	chlist.nsel is set to zero. The method sends an LG_DRAW message and returns WN_KEY_CHANGED
W_KEY_END	chlist.nsel is set to the last item in the choice list. The method sends an LG_DRAW message and returns WN_KEY_CHANGED

W_KEY_TAB	an instance of LISTBOX is created, initialised and made visible (using the utility function hInitVis. The initialisation uses an IN_LISTBOX struct that is set up as indicated in the following code fragment:
	`IN_LISTBOX lst;`
	`lst.flags=IN_LISTBOX_KEEP_ARRAY|IN_LISTBOX_CUR_SET|`
	`IN_LISTBOX_POS_ALIGN_Y|IN_LISTBOX_AUTO_SIZE;`
	`if (self->chlist.matcher)`
	`lst.flags|=IN_LISTBOX_MATCHER;`
	`if (self->chlist.flags&PR_CHLIST_FIXED_WIDTH)`
	`{ /* fixed width code not present for Series 3 */`
	`lst.minwid=p_send2(self,O_LG_SENSE_WIDTH);`
	`lst.flags|=IN_LISTBOX_FIXED_WIDTH;`
	`}`
	`wInquireWindowOffset(0,self->win.id,&lst.pos);`
	`lst.pos.x+=self->lodger.offset.x;`
	`lst.pos.y+=self->lodger.offset.y;`
	`lst.current=self->chlist.nsel;`
	If successfully created and initialised, the LISTBOX handle is written to chlist.pop. The method returns WN_KEY_ABSORB_ON to instruct the dialog box to direct all future key presses to the wn_key method of this control.

any other non-printable key, except W_KEY_DELETE_LEFT	the method sends an LG_DRAW message and returns WN_KEY_NO_CHANGE.
any other key	if chlist.matcher is not NULL, this object is sent an IM_KEY message, passing keycode and modifiers. If the return value is IM_NEW_DISPLAY, chlist.nsel is set to the value returned by sending chlist.matcher an IM_SENSE_VAL message. The method then sends an LG_DRAW message and returns WN_KEY_CHANGED. Otherwise the method attempts to make a first letter match between the choice list items and the value of keycode. This matching is performed by cycling forwards from the current item (so that repeated pressing of an alphabetic key will select, in turn, each item whose text starts with that letter). The method then sends an LG_DRAW message and returns either WN_KEY_NO_CHANGE, or WN_KEY_CHANGED if the first letter match changed the current item

This message is sent from the DLGBOX wn_key method.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the choice list control, assuming the existence of an appropriate graphics context.

If the choice list contains no items, the method simply clears the rectangular area occupied by the control.

Otherwise, draw (using gPrintBoxText) the text of the current item and, if win.flags contains PR_WIN_EMPHASISED, enclosing left and right arrow characters. If the choice list supports incremental

matching the incremental matching cursor is drawn, provided the choice list is emphasised, `chlist.flags` does not include `PR_CHLIST_SUSPENDED` and a pop-out expanded view is not currently displayed.

This message is sent from the `DLGBOX` `wn_draw` method.

WN_SET — Set current item and/or data

```
VOID wn_set(SE_CHLIST *set);
```

Set the data array that contains the choice list items and/or the currently selected item from the `SE_CHLIST` struct pointed to by `set`.

If `set->set_flags` contains `SE_CHLIST_DATA`, the choice list is set to use the data array whose handle is stored in `set->data`. Provided `chlist.flags` does not contain `PR_CHLIST_RETAIN`, any existing `chlist.data` is sent a `DESTROY` message before the handle in `set->data` is copied to `chlist.data`. If `chlist.matcher` is not `NULL`, the value of `set->data` is also written to `chlist.matcher->vmatcher.va`.

If `set->set_flags` contains `SE_CHLIST_NSEL`, the choice list is set to the item with index number `set->nsel` or, if this number is too large, the last item in the choice list. If `chlist.matcher` is not `NULL`, `chlist.matcher` is sent an `IM_SET_VAL` message, passing the value of `chlist.nsel`, and `chlist.matchlen` is set to zero.

If `set->set_flags` contains `SE_CHLIST_RETAIN`, the flag `PR_CHLIST_RETAIN` is ORed into `chlist.flags`, so that the instance of `VARES` whose handle is in `chlist.data` will not subsequently be destroyed. Note that a means of clearing this flag, once set, is not supplied.

On conclusion, the method sends an `LG_DRAW` message.

This message is sent from the `DLGBOX` `wn_set` method, normally invoked from application code by means of the `hDlgSet`, `hDlgSetChlist` and `hDlgSetChlistOn` utility functions.

WN_SENSE — Sense data and selected item

```
VOID wn_sense(SE_CHLIST *set);
```

Writes `chlist.nsel` and `chlist.data` to `set->nsel` and `set->data` respectively.

This message is sent from the `DLGBOX` `wn_sense` method, normally invoked from application code by means of the `hDlgSense` and `hDlgSenseChlist` utility functions.

LG_SENSE_WIDTH — Sense required width

```
INT lg_sense_width(VOID);
```

Returns the width required to display the widest item in the choice list. If there are no items, the method returns the width required to display text of zero length.

This message is sent from the `DLGBOX` `dl_set_size` method.

WN_EMPHASISE — Set emphasis

```
VOID wn_emphasise(INT flags);
```

If `chlist.pop` is not `NULL`, simply send the `WN_EMPHASISE` message to `chlist.pop`.

Otherwise, set or clear `PR_WIN_EMPHASISED` in `win.flags`, depending on whether the passed `flags` value is `TRUE` or `FALSE` and, if `FALSE`, erase any incremental matching cursor. Then send an `LG_DRAW` message.

This message is sent from the `DLGBOX` `dl_take_focus` method.

The NCHLIST numeric choice list class

WIN	LODGER	CHLIST	NCHLIST
flags id	landlord offset width	data flags matcher matchlen nsel pop	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ lg_draw lg_self_check lg_set_id_pos ~~wn_init~~ wn_visible ~~*lg_sense_width*~~ *lg_update*	destroy lg_sense_width wn_draw wn_emphasise ~~wn_init~~ wn_key ~~wn_sense~~ ~~wn_set~~	wn_init wn_sense wn_set

The NCHLIST *class is not present on the Series 3.*

The NCHLIST class provides a choice list control that allows the selection of a positive integer value in the contiguous range from 1 to a specified maximum value.

Class definition

Defined in sub-category file *nchlist.cl* (generated header file *nchlist.g*).

```
CLASS nchlist chlist
Numeric choice list class
    {
    REPLACE wn_init
    REPLACE wn_set
    REPLACE wn_sense
    }
```

Property

There is no property associated with NCHLIST.

NCHLIST methods

WN_INIT — Initialise

```
VOID wn_init(IN_CHLIST *init, PR_WIN *landlord);
```

Initialise the numeric choice list.

The method sets `lodger.landlord` to the passed value of `landlord`. amd creates an instance of the VANUMBER class, storing its handle in `chlist.data`.

The list of numeric choices is not set up during initialisation, the init parameter being ignored. In consequence, there must be a following call to the `wn_set` method.

WN_SET — Set current item and/or maximum value

```
VOID wn_set(SE_NCEDIT *pset);
```

Set the currently selected item and/or the maximum selectable value from the SE_NCEDIT struct pointed to by set. The SE_NCEDIT struct is defined in *ncedit.g* as:

```
typedef struct
    {
    UWORD value;
    UWORD low;
    UWORD high;
    UWORD flags;
    } SE_NCEDIT;
```

For further explanation, see the description of the NCEDIT class, in the *Numeric Editors* chapter.

If pset->flags contains SE_NCEDIT_HIGH, the maximum selectable value is set by sending the component instance of VANUMBER a VAN_SET_MAX message, passing the value of pset->high. It is essential that an instance of NCHLIST receives at least one WN_SET message to set the maximum value after processing a WN_INIT message.

If pset->flags contains SE_NCEDIT_VALUE, the current selection (as specified by chlist.nsel) is set to the item with index number pset->value - 1 and NCHLIST sends itself an LG_DRAW message.

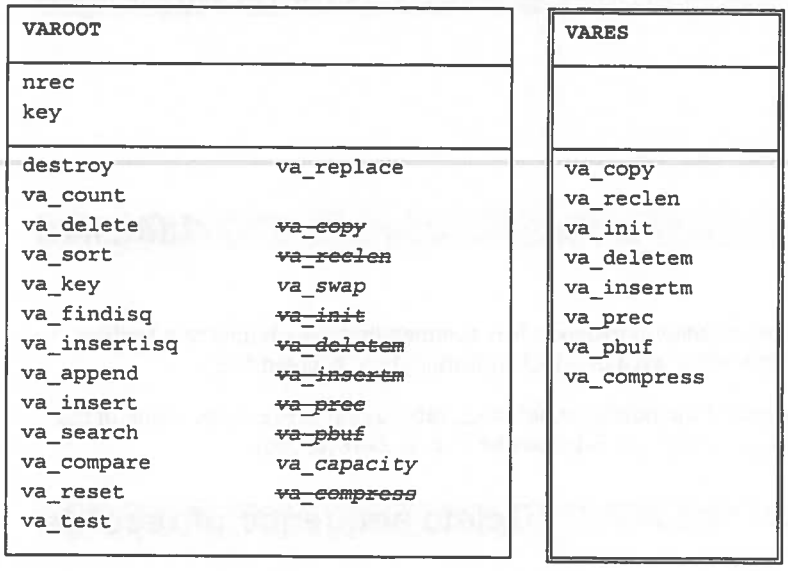
It is the programmer's responsibility to ensure that a WN_SET message does not attempt to set a current selection that exceeds the current maximum value.

WN_SENSE — Sense selected item

```
VOID wn_sense(UWORD *psense);
```

Writes the value of chlist.nsel + 1 to *psense.

The VARES resource array class



The VARES class subclasses VAROOT to provide an array suitable for storing a sequence of leading byte counted text elements, the array being preceded by a byte count of the number of elements in the array. It is primarily intended for storing data such data that has been loaded from a resource file (for example, a MENU resource that contains the set of text items to be displayed in a choice list). It is used as a component by the CHLIST, ACLIST and SMACLIST classes.

Class definition

Defined in sub-category file *vares.cl* (generated header file *vares.g*).

```
CLASS vares varoot
    {
    REPLACE va_copy
    REPLACE va_reclen
    REPLACE va_init
    REPLACE va_deletem
    REPLACE va_insertm
    REPLACE va_prec
    REPLACE va_pbuf
    REPLACE va_compress
    PROPERTY
        {
        UBYTE *data;
        }
    }
```

Property

`vares.data`	Either `NULL` or a pointer to an allocated cell that contains a byte count of the number of elements, followed by the elements, each consisting of leading byte counted text.

VARES methods

VA_COPY — Copy a record

```
UWORD va_copy(UWORD num, UBYTE *prec);
```

Copy the data of record number `num`, not including its leading count byte, to the buffer pointed to by `prec` and return the length of the copied data.

VA_RECLEN — Get record length

```
UWORD va_reclen(UWORD num);
```

Return the length of record number `num`.

The record is located by sending a `VA_PREC` message.

VA_INIT — Initialise

```
VOID va_init(UBYTE *data);
```

Initialise the array to contain the elements pointed to by `data`. It is assumed that `data` points to a leading byte that contains a count of the following items, each of which is leading byte counted text.

Copies the pointer `data` to `vares.data` and, if the pointer is not `NULL`, sets `varoot.nrec` to the value in the first byte pointed to by `data`. Finally, `varoot.key.ofs` is set to one - i.e. `sizeof(UBYTE)`.

VA_DELETEM — Delete sequence of records

```
VOID va_deletem(UWORD num, UWORD nrec);
```

Delete `nrec` records, starting with record number `num`.

The method does nothing if `vares.data` is `NULL`. Otherwise the specified items are removed and the alloc cell adjusted in size by means of a call to `p_adjust`. The value of `varoot.nrec` and the item count in the first byte of the data are adjusted accordingly.

VA_INSERTM — Insert sequence of records

```
VOID va_insertm(UWORD num, UBYTE *prec, UWORD nrec);
```

Insert `nrec` records from the buffer pointed to by `prec`, immediately before record number `num`. The data pointed to by `prec` is assumed to be one or more leading byte counted text items, with no leading item count.

If `vares.data` is `NULL`, the method creates an alloc cell to contain the data, otherwise a gap is opened in the existing cell by means of a call to `p_adjust`.

If this succeeds, the data is copied into the alloc cell. The value of `varoot.nrec` and the item count in the first byte of the data are adjusted accordingly.

The method does nothing to the data and calls `p_leave` if there is insufficient memory to create or expand the alloc cell.

VA_PREC — Point to record

```
UBYTE *va_prec(UWORD num);
```

Return a pointer to record number `num`, including its leading count byte.

VA_PBUF — Point to record data

```
UBYTE *va_pbuf(UWORD num);
```

Return a pointer to the data of record number `num`, that is, excluding its leading count byte.

VA_COMPRESS — Compress

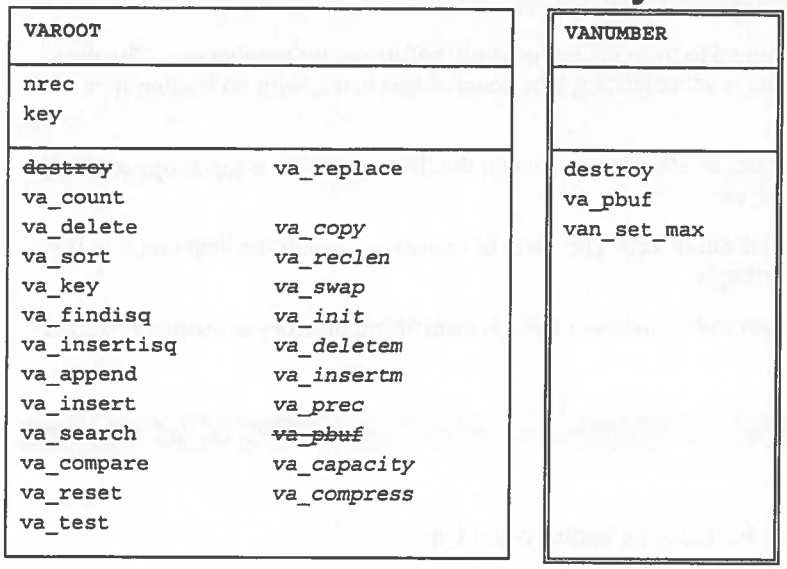
```
VOID va_compress(VOID);
```

Free unused allocated memory.

The method does nothing unless `varoot.nrec` is zero (inserting or deleting records always adjusts the size of the cell accordingly).

If `varoot.nrec` is zero, the method frees the alloc cell pointed to by `vares.data` and sets `vares.data` to be `NULL`.

The VANUMBER numeric array class



The VANUMBER *class is not present on the Series 3.*

The VANUMBER class subclasses VAROOT to provide a pseudo-array that behaves as though it contains records that consist of string representations of consecutive positive integers, starting with "1". It is used as a component by the NCHLIST class.

Class definition

Defined in sub-category file *nchlist.cl* (generated header file *nchlist.g*).

```
CLASS vanumber varoot
    {
    REPLACE destroy=root_destroy
    REPLACE va_pbuf
    ADD van_set_max
    PROPERTY
        {
        TEXT data[8];
        }
    }
```

Property

vanumber.data	A buffer containing the string representation of the array 'record' last referenced in a VA_PBUF message.

VANUMBER methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

This method calls root_destroy since, exceptionally, a VANUMBER array has no allocated memory to contain its 'records'.

VA_PBUF — Point to record data

```
TEXT *va_pbuf(INT num);
```

Return a pointer to 'record' number `num`.

The method uses a call to `p_itob` to generate, in `vanumber.data`, a zero-terminated string representing the number `num` and returns a pointer to this string.

VAN_SET_MAX — Set maximum value

```
VOID van_set_max(INT max);
```

Set the maximum displayable value.

The method simply sets `varoot.nrec` to the value `max`.

CHAPTER 9

NUMERIC EDITORS

This chapter describes the numeric editor dialog components provided by the HWIM library. There are currently seven variants as follows:

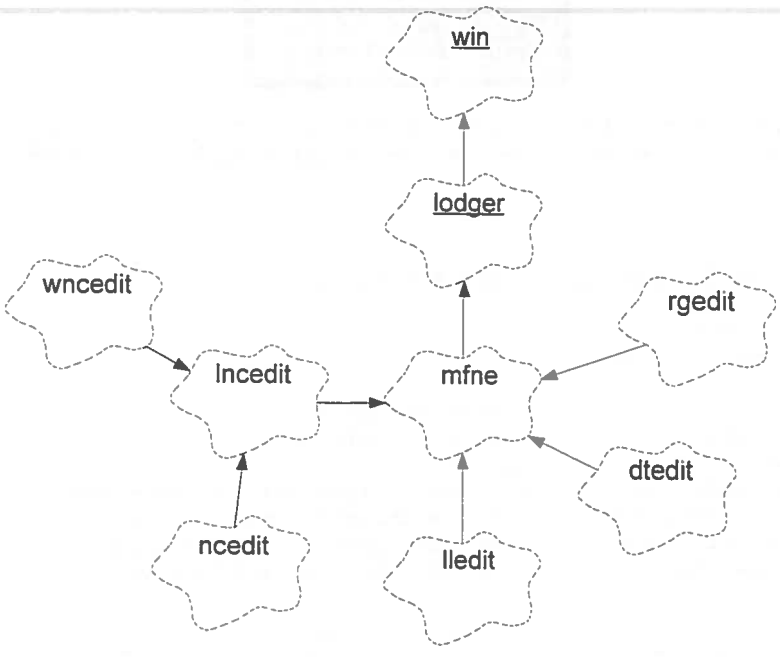
- the multi-field numeric editor that provides the base class for the remaining six variants: it is not intended to be used as a stand-alone class
- the long numeric editor that allows the user to edit a signed long value
- the integer numeric editor that allows the user to edit an unsigned integer value
- the word numeric editor that allows the user to edit a word value
- the date/time numeric editor that allows the user to edit either a date, a time or a duration
- the latitude/longitude editor that allows the user to edit either a latitude or a longitude
- the range numeric editor that allows the user to edit two unsigned words specifying the upper and lower values of a range.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the WIN and LODGER classes described in the *Windows* chapter of the *HWIM Reference* manual.
- the description of the DLGBOX class in the *Dialog Boxes* chapter, particularly the `wn_key` method which may send WN_KEY messages to a dialog box component.

Class diagram



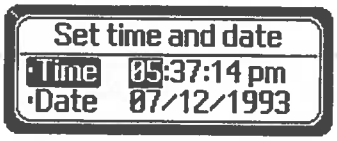
MFNE

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ *wn_sense* ~~wn_draw~~ ~~wn_init~~	destroy wn_init wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*

MFNE
selected cField nField totWidth cPos changed eStr dStr trail f
wn_key wn_draw wn_set wn_emphasise lg_sense_width lg_self_check mf_range_beep

The MFNE multi-field numeric editor class is intended to be subclassed to create a numeric editor control. In addition to the subclasses supplied in the HWIM library, two example subclasses are described later in this chapter.

MFNE supports the display of a series of fields, each of which may be edited, as shown in the following picture:



In this illustration, the *Time* control allows the user to edit the hours, minutes and seconds fields, and to type a *P* or an *A* to toggle the *am/pm* indicator. The *Date* control allows the user to edit the day, the month and the year.

Class definition

Defined in sub-category file *mfne.cl* (generated header file *mfne.g*).

```
CLASS   mfne  lodger
Multi field numeric editor
    (
    REPLACE wn_key              process key press
    REPLACE wn_draw             display current number
    REPLACE wn_set              set data
    REPLACE wn_emphasise        set initial highlight, validate number
    REPLACE lg_sense_width      return width of max number plus cursor
    REPLACE lg_self_check       check the current number is valid
    ADD mf_range_beep           beep and give range info message
```

```
CONSTANTS
    {
    MFNE_MAX_FIELDS 4
    MFNE_MAX_WIDTH 10
    MFNE_MAX_TRAILER 3
    MFNE_LOWER 0
    MFNE_UPPER 1
    MFNE_CURSOR_WIDTH 3
    MFNE_SIGNED 0x01
    MFNE_LEFT_ALIGN 0x02
    MFNE_SUPPRESS_LEADING 0x04
    MFNE_SUPPRESS_SEPARATOR 0x08
    MFNE_TRAILER 0x10
    MFNE_SETS_NEXT 0x20
    MFNE_SETS_PREV 0x40
    MFNE_AUTO_SIZE 0x80
    }

TYPES
    {
    typedef struct
        {
        WORD flags;
        LONG value;
        UBYTE width;
        UBYTE hWidth;
        UBYTE hPos;
        UBYTE separator;
        LONG limits[2];
        } MFNE_FIELD;
    }

PROPERTY
    {
    WORD selected;                                  Current state
    WORD cField;                                    The current field
    WORD nField;                                    The number of fields
    WORD totWidth;                                  The total field width (chars)
    WORD cPos;                                      The cursor position (chars)
    WORD changed;                                   The field has changed
    TEXT eStr[MFNE_MAX_WIDTH+2];                    The current edit context
    TEXT dStr[MFNE_MAX_FIELDS*(MFNE_MAX_WIDTH+1)+MFNE_MAX_TRAILER+2];
    TEXT trail[2][MFNE_MAX_TRAILER+1];              The trailer text
    MFNE_FIELD f[MFNE_MAX_FIELDS];                  The fields
    }
}
```

Property

`mfne.selected`	`TRUE` if the current field is selected/highlighted
`mfne.cField`	the index of the current field: the first field has index zero
`mfne.nField`	the number of fields: less than or equal to `MFNE_MAX_FIELDS`
`mfne.totWidth`	the width of the lodger window: a subclasser need not set this
`mfne.cPos`	the cursor position within the current field - units of characters: a subclasser need not set this
`mfne.changed`	`TRUE` if one or more fields have been changed. This property is set by the `lg_self_check` and `wn_key` methods.
`mfne.eStr`	the editable text of the current field: a subclasser need not set this
`mfne.dStr`	the formatted string representing the entire content of the editor: a subclasser need not set this
`mfne.trail`	the two zero terminated strings that define the trailer text

`mfne.f` an array of `MFNE_FIELD` structs.

The `MFNE_FIELD` struct is defined as follows:

```
typedef struct
    {
    WORD flags;
    LONG value;
    UBYTE width;
    UBYTE hWidth;
    UBYTE hPos;
    UBYTE separator;
    LONG limits[2];
    } MFNE_FIELD;
```

The significance of the members of the `MFNE_FIELD` struct is as follows:

`flags` may contain an ORed combination of the following flags:

`MFNE_TRAILER` which specifies that one of the zero terminated strings specified in `mfne.trail` is to be drawn: the string to be drawn is specified by the current value.

`MFNE_SUPPRESS_SEPARATOR` which specifies that the separator character is to be ignored.

`MFNE_SIGNED` which specifies that the editable value can assume negative values.

`MFNE_SETS_NEXT` which specifies that the `value` member provides a lower bound for the next field.

`MFNE_SETS_PREV` which specifies that the `value` member provides an upper bound for the previous field.

`MFNE_AUTO_SIZE` which specifies that automatic sizing is allowed.

`MFNE_LEFT_ALIGN` which specifies that the field is to be drawn as left aligned text. Note that `MFNE_LEFT_ALIGN` takes precedence over `MFNE_SUPPRESS_LEADING`.

`MFNE_SUPPRESS_LEADING` which specifies that the field is to be drawn without leading.

`value` either the numeric editable value or the index of the trailer text

`width` the character width of the field ignoring the separator

`hWidth` the pixel width of the field in pixels ignoring the separator: a subclasser need not set this

`hPos` the horizontal distance in pixels of the field from the left edge of the lodger window: a subclasser need not set this

`separator` the optional separator character that appears to the right of the editable value/trailer text: the separator is ignored if `MFNE_SUPPRESS_SEPARATOR` is set in `flags`

`limits` the upper and lower limits for the `value` member: the elements should be accessed using the `MFNE_UPPER` and `MFNE_LOWER` symbolic constants. For a trailer field these should be set to one and zero respectively.

MFNE methods

WN_KEY — Handle key press

```
INT wn_key(INT code,INT modifiers);
```

Handle a keypress.

The return value will be WN_KEY_NO_CHANGE whenever:

- the editable value is selected and the keypress is either a left arrow, a right arrow, a tab or a space. When there is more than one field the effect of any of these keys is move the current field, and hence the highlight, to the left, or to the right, by one field.
- the MFNE_SIGNED flag is not set for the *first* field, the editable value is not selected (mfne.selected is FALSE), the cursor is not in the leftmost position, and the keypress is a plus or a minus key. The keypress has no effect.
- the MFNE_SIGNED flag is not set for the *first* field, the editable value is selected, the MFNE_SIGNED field flag is not set in the current field and the keypress is a plus or a minus key. The keypress has no effect.

The return value will be WN_KEY_CHANGED whenever:

- there is only one field, the editable value is not selected and the keypress is a right arrow, a left arrow, a tab or a space. The effect of any of these keypresses is to select the editable value thus highlighting the field.
- there is only one field and the keypress is a full stop or a comma. If the editable value is selected the effect of the full stop (comma) keypress is to increment (decrement) the current value of the current field. Otherwise the effect of either keypress is to select the editable value thus highlighting the current field.
- the current field is a trailer field and the keypress corresponds to the first character of one of the two alternative trailer text strings. The effect of the keypress is to toggle the trailer text.

The return value will be WN_KEY_CHANGED_DEFER whenever:

- there is more than one field, the editable value is selected, and the keypress is a right arrow, a left arrow, a tab or a space. The effect of the key is to move the current field by one field to the right (or to the left in the case of the left arrow).
- there is more than one field and the keypress is a full stop or a comma. The effect of the key is to increment or decrement respectively the current value: if the result is out of range then it is reset to the nearest limit.
- the current field is not selected, the cursor is currently in the leftmost position (cPos is zero) and the keypress is the plus key or the minus key. Sets the sign of the value member of the current field according to the keypress.
- the current field is selected, does not have the MFNE_SIGNED flag set in its flags member, the keypress is a plus or minus key, and the MFNE_SIGNED flag is set in the flag member of the *first* field. The effect of the keypresses is to set the sign of the value member of the *first* field according to the keypress.
- the current field is not selected, the cursor is not in the leftmost position (cPos is not zero), the key press is a plus or a minus, and the MFNE_SIGNED flag is set in the flag member of the *first* field. The effect of the keypresses is to set the sign of the value member of the *first* field according to the keypress.
- the current field is not selected and the keypress is either delete left (W_KEY_DELETE_LEFT) or delete right (W_KEY_DELETE_RIGHT). Unless the cursor position is zero, the character at the current position will be replaced with a numeric space, and the cursor positon decremented by one. Note that for a trailer field the cursor position is always zero.
- the current field is selected and the keypress is either delete left (W_KEY_DELETE_LEFT) or delete right (W_KEY_DELETE_RIGHT). The effect of either keypress is to reset the current field index to the preceding field (mfne.cField is decremented), or, if the current field index is zero, reset the

current field index to the rightmost field. If the cursor position is not zero, the character at the current cursor position is replaced by a numeric space and the cursor positon is decremented.

Note that the WN_KEY_CHANGED_DEFER flag includes the WN_KEY_CHANGED flag: thus WN_KEY_CHANGED_DEFER&WN_KEY_CHANGED would give WN_KEY_CHANGED.

Note also that the while the control key is held down the full stop and comma keys increment/decrement by ten units rather than one.

In all cases the mfne.changed property is set to TRUE if the keypress modified the content or the appearance of the editable value.

WN_DRAW — Draw current string

```
VOID wn_draw(VOID);
```

Draw the current editable value.

If the PR_WIN_EMPHASISED flag is set in win.flags, and mfne.selected is TRUE, draws the current field as white text on a black background.

If the PR_WIN_EMPHASISED flag is set in win.flags, and mfne.selected is not TRUE, draws a cursor in the current field.

Otherwise draws the editable value with no cursor and no highlight.

Note that the text string pointed to by mfne.dStr is not updated before it is drawn. To update and draw the text string the lg_self_check method should be called instead.

WN_SET — Set data

```
VOID wn_set(VOID);
```

Set mnfe.selected to TRUE, make a direct call to the lg_self_check method with an argument of TRUE and evaluate mnfe.totWidth.

The method calls p_leave with an argument of E_GEN_ARG if the lower limit of any field exceeds its upper limit.

WN_EMPHASISE — Emphasise

```
VOID wn_emphasise(UINT flag);
```

If flag is TRUE emphasize the control, otherwise de-emphasise the control.

If flag is TRUE, emphasises the control by ORing PR_WIN_EMPHASISED into win.flags and setting the current field index to zero.

Otherwise, clears PR_WIN_EMPHASISED in win.flags, and erases the cursor.

In both cases an LG_DRAW message is sent before the method returns.

LG_SENSE_WIDTH — Return minimum width

```
INT lg_sense_width(VOID);
```

Return the width in pixels required to draw the lodger window.

The code is effectively:

```
return(mnfe.totWidth*SYSTEM_FONT_NUM_WIDTH);
```

LG_SELF_CHECK — Validate the current number

```
INT lg_self_check(INT can_defer);
```

Validate each field and then update and draw the text string representing the editable value.

The following conditions must be satisfied:

- the `value` member of each field must lie within the specified limits. A `value` member that is out of range is reset to the nearest limit.
- the `value` member of each field must be greater than or equal to the `value` member of the previous field whenever that field has the `MFNE_SETS_NEXT` flag set in its `flags` member. A `value` member that fails this condition is reset to the `value` member of the previous field.
- the `value` member of each field must be less than or equal to the `value` member of the next field whenever that field has the `MFNE_SETS_PREV` flag set in its `flags` member. A `value` member that fails this condition is reset to the `value` member of the next field.

If the above conditions are satisfied and `mfne.changed` is `FALSE`, the method returns `LG_CHECK_OK`.

If the above conditions are satisfied and `mfne.changed` is `TRUE`, the method returns `LG_CHECK_OK_CHANGED`.

If one or more of the above conditions is not satisfied, and `can_defer` is `TRUE`, the method returns `LG_CHECK_OK_CHANGED`.

Otherwise if one or more of the above conditions is not satisfied, the method makes a direct call to the `mf_range_beep` method and then returns `LG_CHECK_FAILED_CHANGED`.

In all cases the method updates and draws the text string representing the editable value.

MF_RANGE_BEEP — Show range information message

```
VOID mf_range_beep(VOID);
```

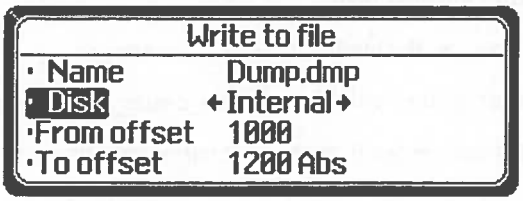
Give a warning beep and present the following message - "Out of range - reset to limit".

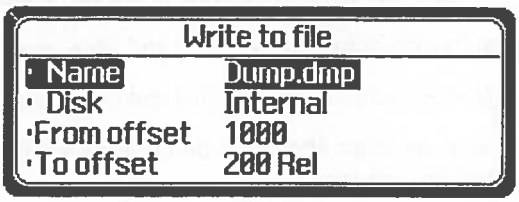
MFNE subclass examples

A basic subclass

The DEMO class is provided as a simple illustration of the subclassing of the MFNE class. It is typical of the numeric editor classes described later in this chapter.

The DEMO control presents an offset followed by three characters that indicate whether the offset is relative or absolute. An example of a DEMO control used as a dialog component is shown in the following pictures:





The DEMO class is defined in a category file as follows:

```
CLASS demo mfne
    {
    ADD wn_init
    REPLACE wn_set
    REPLACE wn_sense
    }
```

The initial appearance of the control is specified in the resource file by the following control resource:

```
RESOURCE CONTROL demo_control_res   /* demo control resource */
    {
    flags=DLGBOX_ITEM_APPL_CAT;
    class=C_DUMO;
    prompt="To offset";
    info=LNCEDIT
        {
        low=0;
        high=65535;
        current=10;
        };
```

(The LNCEDIT struct is used for convenience.)

The fields are initialised and property set in the wn_init method as follows (the wn_init method is called from within the DLGBOX class, before the dialog dl_dyn_init method):

```
METHOD VOID demo_wn_init(PR_DEMO *self,IN_LNCEDIT *pin_lncedit,PR_WIN *landlord)
    {
    /* general settings */
    self->lodger.landlord=landlord;
    self->mfne.selected=FALSE;
    self->mfne.cField=1;
    self->mfne.nField=2;

/* in a 'real' application, trailing text would be read from a resource file */
    p_scpy(&self->mfne.trail[0][0],"Abs");
    p_scpy(&self->mfne.trail[1][0],"Rel");

    /* settings for first field */
    self->mfne.f[0].flags=MFNE_LEFT_ALIGN;
    self->mfne.f[0].value=pin_ncedit->value;
    self->mfne.f[0].limits[MFNE_LOWER]=pin_ncedit->low;
    self->mfne.f[0].limits[MFNE_UPPER]=pin_ncedit->high;
    self->mfne.f[0].width=5;
    self->mfne.f[0].separator=' ';
```

```
    /* settings for second field */
    self->mfne.f[1].flags=MFNE_TRAILER|MFNE_SUPPRESS_SEPARATOR|MFNE_SUPPRESS_LEADING;
    self->mfne.f[1].value=0;
    self->mfne.f[1].limits[MFNE_LOWER]=0;
    self->mfne.f[1].limits[MFNE_UPPER]=1;
    self->mfne.f[1].width=3;

    p_supersend2(self,O_WN_SET);
    }
```

The contents of the control are set and sensed as follows:

```
METHOD VOID demo_wn_set(PR_DEMO *self,SE_DEMO *pset)
    {
    if (pset->flags&SE_DEMO_VALUE_LONG)
        mfne.f[0].value=pset->value_long;
    if (pset->flags&SE_DEMO_VALUE_TRAILER)
        mfne.f[1].value=pset->value_trailer;
    p_supersend2(self,O_WN_SET);
    }

METHOD VOID demo_wn_sense(PR_DEMO *self,SE_DEMO *psense)
    {
    psense->value_long=mfne.f[0].value;
    psense->value_trailer=mfne.f[1].value;
    }
```

where the SE_DEMO struct is defined as follows:

```
typedef struct
    {
    LONG value_long;
    LONG value_trailer;
    }
```

Extended range date editor control

The DATE class is provided as an illustration of the subclassing of the MFNE class and is somewhat more complex than the DEMO class described above.

The DATE control presents an editable date consisting of the day, the month and the year and improves on the DTEDIT control by providing a wider range of allowed dates. Whereas DTEDIT is restricted to an earliest displayable date of 01/01/1980, the DATE class can display dates in the range 01/01/1900 to 31/12/2154 inclusive. An example of a DATE control used as a dialog component is shown in the following picture:



Note that an application which wishes to use the DATE class *must* declare the class number by including the following line in its *.re* file:

```
_C_DATE
```

As the use of the DATE class is non-trivial, the C SDK disks include an example application, installable into a *\sibosdk\hwimdemo* directory, which uses the DATE class.

The example dialog shown above may be defined using the following resource:

```
RESOURCE DIALOG date_dl_res
    {
    title="Current date";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_DATE;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_APPL_CAT;
            }
        };
    }
```

The DLGBOX_ITEM_APPL_CAT flag indicates that the DATE class is supplied by the application.

The DATE class is defined as follows:

```
CLASS date mfne
    {
    REPLACE wn_init
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE lg_self_check
    PROPERTY
        {
        WORD dType;
        UINT day;
        UINT month;
        UINT year;
        ULONG max;
        }
    }
```

The significance of the above property is as follows:

dType — may contain *one* of the following values:

- E_DATE_EUROPE - specifies that the date is displayed as day, month and year.
- E_DATE_USA - specifies that the date is displayed as month, day and year.
- E_DATE_JAPAN - specifies that the date is displayed as year, month and day.

This property is set by the wn_init method.

day — specifies the index of the field which displays the current day.

month — specifies the index of the field which displays the current month.

year — specifies the index of the field which displays the current year.

max — specifies the maximum allowed date in days elapsed since 1/1/1900. This property is set by the wn_init method.

The fields are initialised and property set in the wn_init method, as follows (the wn_init method is called from within the DLGBOX class, before the dialog dl_dyn_init method):

```
LOCAL_C VOID InitField(PR_DATE *self,UINT i,LONG val,LONG min,LONG max,UBYTE width)
    {
    self->mfne.f[i].value=val;
    self->mfne.f[i].limits[MFNE_LOWER]=min;
    self->mfne.f[i].limits[MFNE_UPPER]=max;
    self->mfne.f[i].width=width;
    }

LOCAL_C VOID DayMonthYear(PR_DATE *self,INT i,INT j,INT k)
    {
    self->date.day=i;
    self->date.month=j;
    self->date.year=k;
    }
```

```
#pragma METHOD_CALL

METHOD VOID date_wn_init(PR_DATE *self,VOID *par,PR_WIN *landlord)
    {
    P_DAYSEC ds;
    P_DATE dt;
    E_CONFIG cfg;

    p_getctd(&cfg);
    self->lodger.landlord=landlord;
    self->mfne.selected=FALSE;
    self->mfne.changed=FALSE;
    self->mfne.cField=0;
    self->mfne.nField=3;
    self->mfne.f[0].separator=cfg.dateSeparator;
    self->mfne.f[1].separator=cfg.dateSeparator;
    self->mfne.f[2].flags=MFNE_SUPPRESS_SEPARATOR;
    self->date.dType=cfg.dateType;
    if (self->date.dType==E_DATE_EUROPE)
        DayMonthYear(self,0,1,2);
    else if (self->date.dType==E_DATE_USA)
        DayMonthYear(self,1,0,2);
    else if (self->date.dType==E_DATE_JAPAN)
        DayMonthYear(self,2,1,0);
    InitField(self,self->date.day,1,1,31,2);
    InitField(self,self->date.month,1,1,12,2);
    InitField(self,self->date.year,1,1900,2154,4);
    dt.year=254;
    dt.month=11;
    dt.day=30;
    dt.hour=dt.minute=dt.second=dt.yrday=0;
    p_dttods(&dt,&ds);
    self->date.max=ds.day;
    }
```

The contents of the control are set and sensed as follows:

```
METHOD VOID date_wn_set(PR_DATE *self,ULONG *par)
    {
    P_DAYSEC ds;
    P_DATE dt;

    ds.day=*par;
    ds.sec=0;
    if (ds.day>self->date.max)
        ds.day=self->date.max;
    p_dstodt(&ds,&dt);
    self->mfne.f[self->date.day].value=dt.day+1;
    self->mfne.f[self->date.month].value=dt.month+1;
    self->mfne.f[self->date.year].value=dt.year+1900;
    p_supersend2(self,O_WN_SET);
    }

METHOD VOID date_wn_sense(PR_DATE *self,ULONG *par)
    {
    P_DATE dt;
    P_DAYSEC ds;

    dt.second=0;
    dt.minute=0;
    dt.hour=0;
    dt.day=self->mfne.f[self->date.day].value-1;
    dt.month=self->mfne.f[self->date.month].value-1;
    dt.year=self->mfne.f[self->date.year].value-1900;
    p_dttods(&dt,&ds);
    *par=ds.day;
    }
```

The validity of the current date is checked as follows:

```
METHOD INT date_lg_self_check(PR_DATE *self,INT can_defer)
    {
    INT ret;
    INT days;

    ret=p_supersend3(self,O_LG_SELF_CHECK,can_defer);
    if ((can_defer==FALSE)&&(ret==LG_CHECK_OK_CHANGED))
        {
        days=p_dayinm((INT)self->mfne.f[self->date.year].value-1900,
            (INT)self->mfne.f[self->date.month].value-1);
        if (self->mfne.f[self->date.day].value>(LONG)days)
            {
            hBeep();
            self->mfne.cField=self->date.day;
            self->mfne.f[self->date.day].value=(LONG)days;
            p_supersend3(self,O_LG_SELF_CHECK,TRUE);
            hInfoPrint(MONTH_DAY_RESET);
            ret=LG_CHECK_FAILED_CHANGED;
            }
        }
    return(ret);
    }
```

The `MONTH_DAY_RESET` resource used by the `lg_self_check` method is defined as follows:

```
RESOURCE STRING month_day_reset
    {
    str="Day exceeds days in month: reset";
    }
```

The dialog resource shown above may be used with the following `DATEDLG` example dialog class:

```
CLASS datedlg dlgbox
    {
    REPLACE dl_item_new
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

An instance of the `DATE` class is created by the `dl_item_new` method as follows:

```
METHOD VOID *datedlg_dl_item_new(PR_DATEDLG *self,AD_DLGBOX *par)
    {
    return(f_new(CAT_DAT_DAT,par->class));
    }
```

The dialog is dynamically initialised and its contents sensed before closing using the `dl_dyn_init` and `dl_key` methods as follows:

```
METHOD VOID datedlg_dl_dyn_init(PR_DATEDLG *self)
    {
    P_DAYSEC ds;

    p_send4(((PR_DATWIN *)w_ws->wserv.cli)-
>datwin.time,O_TO_SENSE,SENSE_TIME_DAYSEC,&ds);
    p_send4(self,O_WN_SET,1,&ds.day);
    }

METHOD INT datedlg_dl_key(PR_DATEDLG *self,INT index,INT keycode)
    {
    P_DAYSEC ds;
    PR_DATWIN *cli;

    p_send4(self,O_WN_SENSE,1,&ds.day);
    ds.sec=0;
    cli=(PR_DATWIN *)w_ws->wserv.cli;
    f_leave(p_send4(cli->datwin.time,O_TO_SET,SET_TIME_DAYSEC,&ds));
    return(WN_KEY_CHANGED);
    }
```

LNCEDIT

WIN	LODGER	MFNE
flags id	landlord offset width	selected cField nField totWidth cPos changed eStr dStr trail f
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*	wn_key wn_draw ~~wn_set~~ wn_emphasise lg_sense_width lg_self_check mf_range_beep

LNCEDIT
wn_set wn_sense wn_init

The long numeric editor presents an editable value of type LONG as illustrated in the following picture:



A long numeric editor control can be included in a dialog via the appropriate control resource. The following example would be suitable for the dialog illustrated in the above picture:

```
RESOURCE DIALOG demonstration
    {
    title="Demonstration";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        title="Demonstration";
        flags=DLGBOX_NOTIFY_ENTER;
        CONTROL
            {
            class=C_LNCEDIT;
            prompt="LONG";
            info=LNCEDIT
                {
                low=1;
                high=700000;
                current=500000;
                };
            }
        };
    }
```

The LNCEDIT struct is defined in *hwim.rh* as:

```
STRUCT LNCEDIT
    {
    LONG current=0;
    LONG low=0;
    LONG high=10000000;
    }
```

Thus the example long numeric editor control has replaced the values assigned to the `current`, `low` and `high` members.

Class definition

Defined in sub-category file *ncedit.cl* (generated header file *ncedit.g*).

```
CLASS   lncedit  mfne
Single field long numeric editor
    {
    REPLACE wn_set              set data
    REPLACE wn_sense            sense value
    REPLACE wn_init             initialise editor

    CONSTANTS
        {
        SE_LNCEDIT_VALUE          0x01
        SE_LNCEDIT_LOW            0x02
        SE_LNCEDIT_HIGH           0x04
        }

    TYPES
        {
        typedef struct
            {
            LONG value;
            LONG low;
            LONG high;
            } IN_LNCEDIT;
        typedef struct
            {
            LONG value;
            LONG low;
            LONG high;
            UWORD flags;
            } SE_LNCEDIT;
        }
    }
```

Property

There is no property associated with the LNCEDIT class.

LNCEDIT methods

WN_INIT — Initialise editor

```
VOID wn_init(IN_LNCEDIT *init,PR_WIN *landlord);
```

Initialise the long numeric editor from the IN_LNCEDIT struct pointed to by `init`, and write `landlord`, the handle of the owning landlord window, to `lodger.landlord`.

The IN_LNCEDIT struct is defined as follows:

```
typedef struct
    {
    LONG value;
    LONG high;
    LONG low;
    } IN_LNCEDIT;
```

The members of the IN_LNCEDIT struct have the following significance:

value	the initial value for the editable LONG.
high	the upper limit for the editable LONG.
low	the lower limit for the editable LONG.

WN_SET — Set data

```
VOID wn_set(SE_LNCEDIT *set);
```

Set the long numeric editor according to the content of the SE_LNCEDIT struct pointed to by set.

The SE_LNCEDIT struct is defined as follows:

```
typedef struct
    {
    LONG value;
    LONG high;
    LONG low;
    UWORD flags;
    } SE_LNCEDIT;
```

The setting is controlled by ORing one or more of the following flags into the flags member of the SE_LNCEDIT struct

SE_LNCEDIT_VALUE	indicates that the editable value is to be set
SE_LNCEDIT_HIGH	indicates that the upper limit is to be set
SE_LNCEDIT_LOW	indicates that the lower limit is to be set

WN_SENSE — Sense value

```
VOID wn_sense(LONG *sense);
```

Write the current value of the long numeric editor to the LONG pointed to by sense.

NCEDIT

WIN	LODGER	MFNE	LNCEDIT
flags id	landlord offset width	selected cField nField totWidth cPos changed eStr dStr trail f	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*	wn_key wn_draw ~~wn_set~~ wn_emphasise lg_sense_width lg_self_check mf_range_beep	~~wn_set~~ ~~wn_sense~~ ~~wn_init~~

NCEDIT
wn_set wn_sense wn_init

The unsigned word numeric editor presents an editable value of type UWORD as illustrated in the following picture:



An unsigned word numeric editor control can be included in a dialog via the appropriate control resource. The following example would be suitable for the dialog illustrated in the above picture:

```
RESOURCE DIALOG demonstration
    {
    title="Position to-do entry";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_NCEDIT;
            prompt="Position within list";
            info=NCEDIT
                {
                low=1;
                high=100;
                current=1;
                };
            }
        };
    }
```

The NCEDIT struct is defined in *hwim.rh* as:

```
STRUCT NCEDIT
    {
    UWORD current=0;
    UWORD low=0;
    UWORD high=65535;
    }
```

Thus the example unsigned word numeric editor control (see above) has replaced the values assigned to the current, low and high members.

Class definition

Defined in sub-category file *ncedit.cl* (generated header file *ncedit.g*).

```
CLASS   ncedit  lncedit
Single field unsigned word numeric editor
    {
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_init

    CONSTANTS
        {
        SE_NCEDIT_VALUE       0x01
        SE_NCEDIT_LOW         0x02
        SE_NCEDIT_HIGH        0x04
        }

    TYPES
        {
        typedef struct
            {
            UWORD value;
            UWORD low;
            UWORD high;
            } IN_NCEDIT;
        typedef struct
            {
            UWORD value;
            UWORD low;
            UWORD high;
            UWORD flags;
            } SE_NCEDIT;
        }
    }
```

Property

There is no property associated with the NCEDIT class.

NCEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_NCEDIT *init,PR_WIN *landlord);
```

Initialise the unsigned word numeric editor from the IN_NCEDIT struct pointed to by init, and write landlord, the handle of the owning landlord window, to lodger.landlord.

The IN_NCEDIT struct is defined as follows:

```
typedef struct
    {
    UWORD value;
    UWORD low;
    UWORD high;
    } IN_NCEDIT;
```

The members of the IN_NCEDIT struct have the following significance:

value	the initial value for the editable UWORD
high	the upper limit for the editable UWORD
low	the lower limit for the editable UWORD

WN_SET — Set new values

```
VOID wn_set(SE_NCEDIT *set);
```

Set the unsigned word numeric editor according to the content of the SE_NCEDIT struct pointed to by set.

The SE_NCEDIT struct is defined as follows:

```
typedef struct
    {
    UWORD value;
    UWORD high;
    UWORD low;
    UWORD flags;
    } SE_NCEDIT;
```

The setting is controlled by ORing one or more of the following flags into the flags member of the SE_NCEDIT struct:

SE_NCEDIT_VALUE	indicates that the editable unsigned word is to be set
SE_NCEDIT_HIGH	indicates that the upper limit of the editable unsigned word is to be set
SE_NCEDIT_LOW	indicates that the lower limit for the editable unsigned word is to be set

WN_SENSE — Sense new value

```
VOID wn_sense(UWORD *sense);
```

Write the current value of the editable unsigned word to the UWORD pointed to by sense.

WNCEDIT

WIN	LODGER	MFNE	LNCEDIT
flags id	landlord offset width	selected cField nField totWidth cPos changed eStr dStr trail f	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~*lg_sense_width*~~ *lg_update*	wn_key wn_draw ~~wn_set~~ wn_emphasise lg_sense_width lg_self_check mf_range_beep	~~wn_set~~ ~~wn_sense~~ ~~wn_init~~

WNCEDIT
wn_set wn_sense wn_init

The word numeric editor presents an editable value of type WORD as illustrated in the following picture:



A word numeric editor control can be included in a dialog via the appropriate control resource. The following example would be suitable for the dialog illustrated in the above picture:

```
RESOURCE DIALOG demonstration
    {
    title="Montreal";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_WNCEDIT;
            prompt="Temperature";
            info=WNCEDIT
                {
                current=-100;
                };
            }
        };
    }
```

The WNCEDIT struct is defined in *hwim.rh* as:

```
STRUCT WNCEDIT
    {
    WORD current=0;
    WORD low=-32768;
    WORD high=32767;
    }
```

Thus the example word numeric editor control (see above) has replaced the value assigned to the `current` member.

Class definition

Defined in sub-category file *ncedit.cl* (generated header file *ncedit.g*).

```
CLASS   wncedit  lncedit
Single field word numeric editor
    {
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_init

    CONSTANTS
        {
        SE_WNCEDIT_VALUE          0x01
        SE_WNCEDIT_LOW            0x02
        SE_WNCEDIT_HIGH           0x04
        }

    TYPES
        {
        typedef struct
            {
            WORD value;
            WORD low;
            WORD high;
            } IN_WNCEDIT;
        typedef struct
            {
            WORD value;
            WORD low;
            WORD high;
            WORD flags;
            } SE_WNCEDIT;
        }
    }
```

Property

There is no property associated with the `WNCEDIT` class.

WNCEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_WNCEDIT *init,PR_WIN *landlord);
```

Initialise the signed word numeric editor from the `IN_WNCEDIT` struct pointed to by `init`, and write `landlord`, the handle of the owning landlord window, to `lodger.landlord`.

The `IN_WNCEDIT` struct is defined as follows:

```
typedef struct
    {
    WORD value;
    WORD low;
    WORD high;
    } IN_WNCEDIT;
```

The members of the `IN_WNCEDIT` struct have the following significance:

`value`	the initial value for the editable `WORD`
`high`	the upper limit for the editable `WORD`
`low`	the lower limit for the editable `WORD`

WN_SET — Set new values

```
VOID wn_set(SE_WNCEDIT *set);
```

Set the signed word numeric editor according to the content of the SE_WNCEDIT struct pointed to by set.

The SE_WNCEDIT struct is defined as follows:

```
typedef struct
    {
    WORD value;
    WORD high;
    WORD low;
    WORD flags;
    } SE_WNCEDIT;
```

The setting is controlled by ORing one or more of the following flags into the flags member of the SE_WNCEDIT struct:

SE_WNCEDIT_VALUE	indicates that the editable value is to be set
SE_WNCEDIT_HIGH	indicates that the upper limit is to be set
SE_WNCEDIT_LOW	indicates that the lower limit is to be set

WN_SENSE — Sense new value

```
VOID wn_sense(WORD *sense);
```

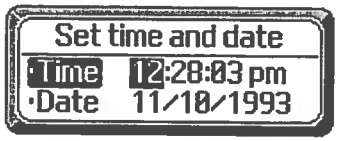
Write the current value of the editable signed word to the WORD pointed to by sense.

DTEDIT

WIN	LODGER	MFNE
flags id	landlord offset width	selected cField nField totWidth cPos changed eStr dStr trail f
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*	wn_key wn_draw ~~wn_set~~ wn_emphasise lg_sense_width ~~lg_self_check~~ mf_range_beep

DTEDIT
cal dType changed inSelfCheck in
wn_set wn_sense wn_init wn_emphasise wn_key lg_self_check

The date/time numeric editor presents a date, a time or a duration as illustrated in the following picture:



A date/time numeric editor control can be included in a dialog via the appropriate control resource. The following example would be suitable for the dialog illustrated in the above picture:

```
RESOURCE DIALOG demonstration
    {
    title="Set time and date";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_DTEDIT;
            prompt="Time";
            info=DTEDIT
                {
                flags=IN_DTEDIT_HHMMSS;
                };
            },
        CONTROL
            {
            class=C_DTEDIT;
            prompt="Date";
            info=DTEDIT
                {
                flags=IN_DTEDIT_DDMMYYYY;
                };
            }
        };
    }
```

The DTEDIT resource struct is defined in *hwim.rh* as:

```
STRUCT DTEDIT
    {
    WORD flags;
    LONG current;
    LONG low;
    LONG high;
    }
```

The example date/time numeric editor controls do not set initial values for the `current`, `low` and `high` members, but merely set appropriate values for `flags`. Note that any supplied values of `current`, `low` and `high` will only be used to initialise the control if the value `IN_DTEDIT_INIT` is set in `flags`. If this value is not present, any values specified in the initialisation resource are ignored and the control is initialised with a set of standard values based on the current time and date.

The format of the display is controlled on initialisation by setting **one** of the following flags into the `flags` member of the DTEDIT struct:

`IN_DTEDIT_DDMMYYYY`	initialise as a date editor to display a date in day, month and year format: the `current` date is specified as days elapsed since 1/1/1900.
`IN_DTEDIT_HHMMSS`	initialise as a time editor to display a time as hours, minutes and seconds. The `current` time is specified in seconds elapsed since midnight.
`IN_DTEDIT_HHMM`	initialise as a time editor to display a time as hours and minutes. The `current` time is specified as seconds elapsed since midnight.
`IN_DTEDIT_HHMMSS_D`	initialise as a time editor to display a duration as hours, minutes and seconds. The `current` duration is specified in seconds.
`IN_DTEDIT_HHMM_D`	initialise as a time editor to display a duration as hours and minutes. The `current` duration is specified in seconds.
`IN_DTEDIT_HHMMSS_ND`	initialise as a time editor to display a negative duration as hours, minutes and seconds. The `current` duration is specified in seconds.
`IN_DTEDIT_HHMM_ND`	initialise as a time editor to display a negative duration as hours and minutes. The `current` duration is specified in seconds.

Note: on the Series 3a version of the DTEDIT class the user may select the required date by pressing the Tab key to obtain a calendar view. (The calendar view is implemented by the CALWIN class.) This is especially convenient for selecting, for example, the first Monday in a given month.

Class definition

Defined in sub-category file *dtedit.cl* (generated header file *dtedit.g*).

On the Series 3a the class definition is as follows:

```
CLASS   dtedit  mfne
Date, Time and Duration editor based on the MFNE.
    {
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_init
    REPLACE wn_emphasise
    REPLACE lg_self_check
    REPLACE wn_key

    CONSTANTS
        {
        SE_DTEDIT_VALUE          0x01
        SE_DTEDIT_LOW            0x02
        SE_DTEDIT_HIGH           0x04

        PR_DTEDIT_DATE           0x0000
        PR_DTEDIT_TIME           0x0100
        PR_DTEDIT_DURATION       0x0200
        PR_DTEDIT_SECONDS        0x1000
        PR_DTEDIT_NEGATIVE       0x2000
        PR_DTEDIT_AMPM           0x4000
        PR_DTEDIT_NEG_DURATION   (PR_DTEDIT_DURATION|PR_DTEDIT_NEGATIVE)

        IN_DTEDIT_INIT           (SE_DTEDIT_VALUE|SE_DTEDIT_LOW|SE_DTEDIT_HIGH)
        IN_DTEDIT_DDMMYYYY       (PR_DTEDIT_DATE)
        IN_DTEDIT_HHMMSS         (PR_DTEDIT_TIME|PR_DTEDIT_SECONDS)
        IN_DTEDIT_HHMM           (PR_DTEDIT_TIME)
        IN_DTEDIT_HHMMSS_D       (PR_DTEDIT_TIME|PR_DTEDIT_DURATION|PR_DTEDIT_SECONDS)
        IN_DTEDIT_HHMM_D         (PR_DTEDIT_TIME|PR_DTEDIT_DURATION)
        IN_DTEDIT_HHMMSS_ND
(PR_DTEDIT_TIME|PR_DTEDIT_NEG_DURATION|PR_DTEDIT_SECONDS)
        IN_DTEDIT_HHMM_ND        (PR_DTEDIT_TIME|PR_DTEDIT_NEG_DURATION)
        }

    TYPES
        {
        typedef struct
            {
            UWORD flags;         type of edit box
            LONG value;
            LONG low;
            LONG high;
            } IN_DTEDIT;
        typedef struct
            {
            UWORD flags;
            LONG value;
            LONG low;
            LONG high;
            } SE_DTEDIT;
        }

    PROPERTY 1
        {
        VOID *cal;
        WORD dType;
        UBYTE changed;
        UBYTE inSelfCheck;
        IN_DTEDIT in;
        }
    }
```

On the Series 3 the class definition differs in that:

- the `wn_emphasise` and `wn_key` methods *are not* replaced.

- the dtedit.cal property is absent.

Property

dtedit.cal	this is either the handle of an instance of the CALWIN class or NULL. The CALWIN object provides a graphical calendar view. Note:on the Series 3 this item of property is absent.
dtedit.dType	specifies the system date format: may be one of either E_DATE_EUROPE for day/month/year, E_DATE_USA for month/day/year or E_DATE_JAPAN for year/month/day.
dtedit.changed	see the description of the wn_sense and lg_self_check methods.
dtedit.inSelfCheck	not used.
dtedit.in	an IN_DTEDIT struct that contains the current editable value, the upper and lower limits and the initialisation flag

DTEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_DTEDIT *init,PR_WIN *landlord);
```

Initialise the date/time numeric editor from the IN_DTEDIT struct pointed to by init, and write landlord, the handle of the owning landlord window, to lodger.landlord. Set dtedit.dType according to the current system setting.

The IN_DTEDIT struct is defined as follows:

```
TYPEDEF STRUCT
    {
    WORD flags;
    LONG current;
    LONG low;
    LONG high;
    } IN_DTEDIT;
```

The members of the IN_DTEDIT struct have the following significance:

flags	a combination of initialisation flags, as described below
current	the initial value for the editable date/time
high	the initial upper limit for the editable date/time
low	the initial lower limit for the editable date/time

The appearance and behaviour of the date/time editor is specified by assigning **one** of the following flags into the flags member (note that the system will automatically display the time in am/pm or 24 hour format according to the current system settings):

IN_DTEDIT_DDMMYYYY	initialise as a date editor to display a date in day, month and year format: the current, low and high dates are specified as days elapsed since 01/01/1900. *Note that the control can not display dates before* 01/01/1980.
IN_DTEDIT_HHMMSS	initialise as a time editor to display a time as hours, minutes and seconds. The current, low and high times are specified in seconds elapsed since midnight. Whole days are ignored, so these values could also be supplied in system time format.
IN_DTEDIT_HHMM	initialise as a time editor to display a time as hours and minutes. The current, low and high times are specified in seconds elapsed since midnight. Whole days are ignored, so these values could also be supplied in system time format.

IN_DTEDIT_HHMMSS_D	initialise as a time editor to display a duration as hours, minutes and seconds. The current, low and high durations are specified in seconds. Whole days are ignored, so these values could also be supplied in system time format.
IN_DTEDIT_HHMM_D	initialise as a time editor to display a duration as hours and minutes. The current, low and high durations are specified in seconds. Whole days are ignored, so these values could also be supplied in system time format.
IN_DTEDIT_HHMMSS_ND	initialise as a time editor to display a negative duration as hours, minutes and seconds. The current, low and high durations are specified in seconds. Whole days are ignored, so these values could also be supplied in system time format.
IN_DTEDIT_HHMM_ND	initialise as a time editor to display a negative duration as hours and minutes. The current, low and high durations are specified in seconds. Whole days are ignored, so these values could also be supplied in system time format.

Any supplied values of current, low and high will only be used to initialise the control if the value IN_DTEDIT_INIT is ORed into flags (in which case all three values must be supplied). If this value is not present, any values specified in the initialisation resource are ignored and the control is initialised with a set of standard values, dependent on the value of flags, as follows:

IN_DTEDIT_DDMMYYYY	The upper and lower limits are set to the dates 01/01/1980 and 31/12/2049 respectively, and the current value is set to today's date.
IN_DTEDIT_HHMMSS IN_DTEDIT_HHMM IN_DTEDIT_HHMMSS_D IN_DTEDIT_HHMM_D	The upper and lower limits are set to the times 00h00m(00s) and 23h59m(59s) respectively, and the current value is set to the time of day.
IN_DTEDIT_HHMMSS_ND IN_DTEDIT_HHMM_ND	The upper and lower limits are set to the times -23h59m(59s) and 23h59m(59s) respectively, and the current value is set to the time of day.

WN_SET — Set new values

```
VOID wn_set(SE_DTEDIT *set);
```

Set the date/time editor according to the content of the SE_DTEDIT struct pointed to by set.

The SE_DTEDIT struct is defined as follows:

```
typedef struct
    {
    UWORD flags;
    LONG value;
    LONG low;
    LONG high;
    } SE_DTEDIT;
```

The property to be set is indicated by ORing one or more of the following flags into the flags field of the above struct.

SE_DTEDIT_VALUE	indicates that the current value is to be set.
SE_DTEDIT_LOW	indicates that the lower limit is to be set.
SE_DTEDIT_HIGH	indicates that the upper limit is to be set.

The interpretation of the values of value, low and high in the SE_DTEDIT struct depends on the type of data that the control has been initialised to display. See the explanation of the IN_DTEDIT_XXX flags in the description of the wn_init method.

WN_SENSE — Sense value

```
VOID wn_sense(SE_DTEDIT *sense);
```

Write the current value of the editable date/time to the value member of the SE_DTEDIT struct pointed to by sense. Note that the method does not write to sense->flags, sense->low or sense->high.

The interpretation of value depends on the type of data that the control has been initialised to display. See the explanation of the IN_DTEDIT_XXX flags in the description of the wn_init method.

If the editable value is a date, and the day is outside the allowed range, resets the day to the last of the month and sets dtedit.changed to FALSE, otherwise sets dtedit.changed to TRUE.

WN_KEY — Handle a keypress

```
VOID wn_key(INT keycode,INT modifiers);
```

Handle a keypress.

If dtedit.cal is non-zero:

- sends the keypress to the calendar view by sending a WN_KEY message with arguments of keycode and modifiers.
- if the keypress cancels the calendar view, i.e. the return value is WN_KEY_CANCELLED, removes the calendar view by sending a DESTROY message to dtedit.cal and then writes zero to dtedit.cal.
- if the keypress selects the current date, i.e the return value is WN_KEY_CHANGED, senses the current date in the calendar view, sets this as the current date in the DTEDIT control, removes the calendar view by sending a DESTROY message to dtedit.cal and then writes zero to dtedit.cal.

Otherwise if keycode is W_KEY_TAB *and* the control is not a time editor - i.e. dtedit.flags does not contain PR_DTEDIT_TIME - *and* w_ws->wserv.flags contains PR_WSERV_FULLSCREEN:

- validates the current date in the DTEDIT control by sending self an LG_SELF_CHECK message and, if the return value is neither LG_CHECK_OK nor LG_CHECK_OK_CHANGED, returns.
- creates an instance of the CALWIN class and writes the handle to dtedit.cal.
- initialises the calendar view by sending a WN_INIT message to dtedit.cal. The calendar view is centred. The current date is set to the current date in the DTEDIT control. The number of months displayed is equal to twelve if modifiers contains W_CTRL_MODIFIER, three if modifiers contains W_SHIFT_MODIFIER, or one otherwise,

Otherwise supersends a WN_KEY message with arguments of keycode and modifiers and returns the return value.

Note: on the Series 3 this method is not replaced.

WN_EMPHASISE — Emphasise

```
VOID wn_emphasise(INT flag);
```

Emphasise the control or calendar view if flag is TRUE, otherwise de-emphasise the control or calendar view.

If dtedit.cal is non-zero, sends a WN_EMPHASISE message to dtedit.cal with an argument of flag.

Otherwise supersends a WN_EMPHASISE message with an argument of flag.

Note: on the Series 3 this method is not replaced.

LG_SELF_CHECK — Validate values and fields

```
INT lg_self_check(VOID);
```

Validate the current date/time and the fields.

Supersends an LG_SELF_CHECK message and return either its return value, or alternatively -1 if either of the following are true:

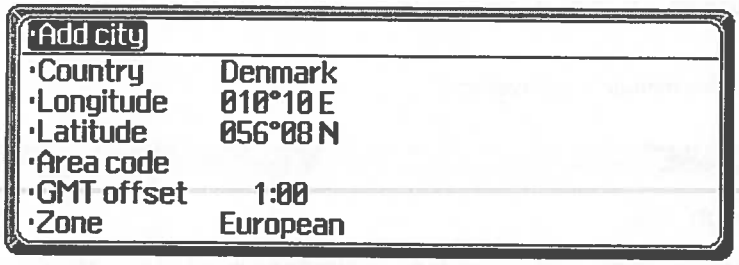
- the editable value is outside of the allowed limits in which case reset to its nearest limit and sends an MF_RANGE_BEEP message.
- dtedit.changed is TRUE in which case sends an MF_RANGE_BEEP message.

LLEDIT

WIN	LODGER	MFNE
flags id	landlord offset width	selected cField nField totWidth cPos changed eStr dStr trail f
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*	wn_key wn_draw ~~wn_set~~ wn_emphasise lg_sense_width lg_self_check mf_range_beep

LLEDIT
wn_set wn_sense wn_init

The latitude/longitude editor presents a latitude or a longitude as illustrated in the following picture:



A latitude/longitude editor control can be included in a dialog via the appropriate control resource. The following example would be suitable for a dialog with a title and two latitude/longitude controls:

```
RESOURCE DIALOG demonstration
    {
    title="Set latitude and longitude";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_LLEDIT;
            prompt="Latitude";
            info=LLEDIT
                {
                flags=IN_LLEDIT_LATITUDE;
                current=3368;
                };
            },
        CONTROL
            {
            class=C_LLEDIT;
            prompt="Longitude";
            info=LLEDIT
                {
                flags=IN_DTEDIT_LONGITUDE;
                current=33968;
                };
            }
        };
    }
```

The `LLEDIT` struct is defined in *hwim.rh* as:

```
STRUCT LLEDIT
    {
    WORD flags;
    WORD value=0;
    }
```

The latitude/longitude `value` is specified in minutes of arc: a positive sign indicating a latitude/longitude in the northern /western hemisphere, and a negative sign indicating a latitude/longitude in the southern /eastern hemisphere. Thus -604 corresponds to a latitude of 10° 4 S or a longitude of 10° 4 E.

The behaviour of a latitude/longitude editor is specified by assigning one of the following flags to the `flags` member.

`IN_LLEDIT_LATITUDE`	display `value` as a latitude.
`IN_LLEDIT_LONGITUDE`	display `value` as a longitude.

Class definition

Defined in sub-category file *lledit.cl* (generated header file *lledit.g*).

```
CLASS   lledit  mfne
Latitude,Longitude editor based on the MFNE.
    {
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_init

    CONSTANTS
        {
    IN_LLEDIT_LATITUDE     0
    IN_LLEDIT_LONGITUDE    1
        }
```

```
TYPES
    {
    typedef struct
        {
        WORD flags;
        WORD value;
        } IN_LLEDIT;
    typedef struct
        {
        WORD value;
        } SE_LLEDIT;
    }
}
```

Property

There is no property associated with the LLEDIT class.

LLEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_LLEDIT *init,PR_WIN *landlord);
```

Initialise the latitude/longitude editor from the IN_LLEDIT struct pointed to by init, and write landlord, the handle of the owning landlord window, to lodger.landlord.

The IN_LLEDIT struct is defined as follows:

```
typedef struct
    {
    WORD flags;
    WORD value;
    } IN_LLEDIT;
```

The behaviour of a latitude/longitude editor is specified by assigning one of the following flags to the flags member:

IN_LLEDIT_LATITUDE	value is to be displayed as a latitude
IN_LLEDIT_LONGITUDE	value is to be displayed as a longitude

Note that the behaviour of a latitude/longitude editor can not be changed once it has been initialised.

The value member of the IN_LLEDIT struct specifies the angle in units of arc minutes. A positive sign indicates that the angle is either a northern latitude, or a western longitude depending on the editors initialisation flag. Otherwise the angle is either a southern latitude or an eastern longitude.

WN_SET — Set new values

```
VOID wn_set(SE_WNCEDIT *set);
```

Set the current angle of the latitude/longitude editor to the value member of the SE_LLEDIT struct which is pointed to by set.

The SE_LLEDIT struct is defined as follows:

```
typedef struct
    {
    WORD value;
    } SE_LLEDIT;
```

See above for a description of the value member.

WN_SENSE — Sense current value

```
VOID wn_sense(SE_LLEDIT *sense);
```

Write the current value of the latitude/longitude editor to the `value` member of the `SE_LLEDIT` struct pointed to by `sense`.

The `SE_LLEDIT` struct is defined as follows:

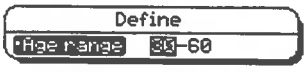
```
typedef struct
    {
    WORD value;
    } SE_LLEDIT;
```

See above for a description of the `value` member.

RGEDIT

WIN	LODGER	MFNE	RGEDIT
flags id	landlord offset width	selected cField nField totWidth cPos changed eStr dStr trail f	in
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*	wn_key wn_draw ~~wn_set~~ wn_emphasise lg_sense_width lg_self_check mf_range_beep	wn_set wn_sense wn_init

The range numeric editor presents two editable unsigned words as illustrated in the following picture:



A range numeric editor control can be included in a dialog via the appropriate control resource. The following example would be suitable for the dialog shown in the above picture:

```
RESOURCE DIALOG demonstration
    {
    title="Define";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_RGEDIT;
            prompt="Age range";
            info=RGEDIT
                {
                value_1=30;
                value_2=60;
                };
            }
        };
    }
```

The RGEDIT struct is defined in *hwim.rh* as:

```
STRUCT RGEDIT
    {
    WORD low=1;
    WORD value_1=1;
    WORD value_2=9999;
    WORD high=9999;
    }
```

where value_1 must be not greater than value_2 and both values must be not less than low, and not greater than high.

Class definition

Defined in sub-category file *rgedit.cl* (generated header file r*gedit.g*).

```
CLASS   rgedit  mfne
Range editor based on the MFNE.
    {
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_init

    CONSTANTS
        {
        SE_RGEDIT_LOW           0x01
        SE_RGEDIT_VALUE_1       0x02
        SE_RGEDIT_VALUE_2       0x04
        SE_RGEDIT_HIGH          0x08
        IX_RGEDIT_LOW           0       /* Indices into value */
        IX_RGEDIT_VALUE_1       1
        IX_RGEDIT_VALUE_2       2
        IX_RGEDIT_HIGH          3
        }

    TYPES
        {
        typedef struct
            {
            UWORD value[4];
            } IN_RGEDIT;
        typedef struct
            {
            UWORD value[4];
            UWORD flags;
            } SE_RGEDIT;
        }

    PROPERTY
        {
        IN_RGEDIT in;
        }
    }
```

Property

`rgedit.in`	not used.

RGEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_RGEDIT *init,PR_WIN *landlord);
```

Initialise the range numeric editor from the `IN_RGEDIT` struct pointed to by `init`, and write `landlord`, the handle of the owning landlord window, to `lodger.landlord`.

The `IN_RGEDIT` struct is defined as follows:

```
typedef struct
    {
     UWORD value[4];
    } IN_RGEDIT;
```

The `value` array is indexed as follows:

`IX_RGEDIT_LOW`	index of lower limit in `value` array.
`IX_RGEDIT_VALUE_1`	index of lower current value in `value` array.
`IX_RGEDIT_VALUE_2`	index of upper current value in `value` array.
`IX_RGEDIT_HIGH`	index of upper limit in `value` array.

WN_SET — Set new values

```
VOID wn_set(SE_RGEDIT *set);
```

Set the current values in the range editor to those specified by the `SE_RGEDIT` struct pointed to by `set`.

The `SE_RGEDIT` struct is defined as follows:

```
typedef struct
    {
    UWORD value[4];
    UWORD flags;
    } SE_RGEDIT;
```

The `value` array is indexed as explained above.

The property to be set is indicated by ORing one or more of the following flags into the flags field of the above struct.

`SE_RGEDIT_LOW`	indicates that the lower limit is to be set.
`SE_RGEDIT_VALUE_1`	indicates that the current lower value is to be set.
`SE_RGEDIT_VALUE_2`	indicates that the current upper value is to be set.
`SE_RGEDIT_HIGH`	indicates that the upper limit is to be set.

WN_SENSE — Sense new value

```
VOID wn_sense(SE_RGEDIT *sense);
```

Write the current values and the limits of the range editor to the `SE_RGEDIT` struct pointed to by `sense`.

The `SE_RGEDIT` struct is defined above.

CHAPTER 10

TEXT EDITORS

This chapter documents the text editor classes. These classes are:

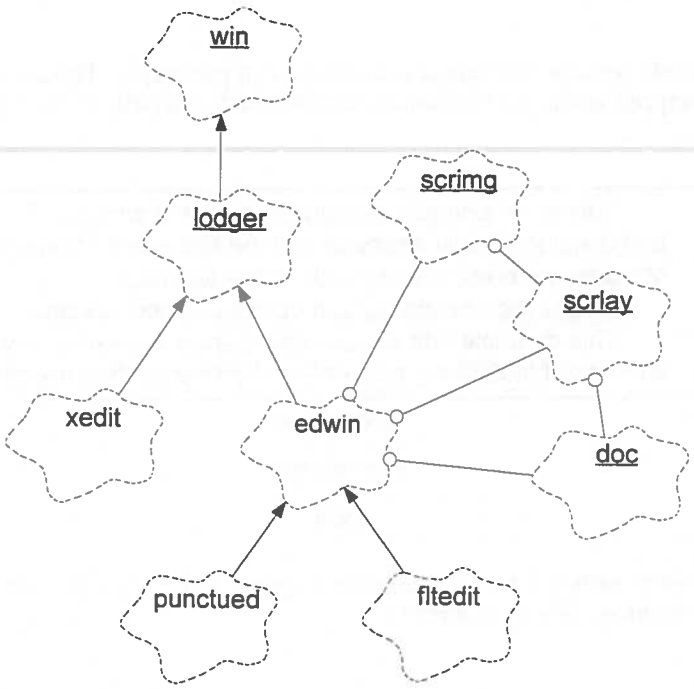
EDWIN	The superclass for the majority of text editor windows. It is a lodger window that provides a wide range of facilities for the formatted display of editable text.
FLTEDIT	A dialog control that allows the user to edit a floating point number.
PUNCTUED	A dialog control that allows the user to select a punctuation character.
XEDIT	A dialog control allowing secret text entry - a password, for example.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the WIN and LODGER classes described in the *Windows* chapter,
- the EPDOC and EPFDOC classes described in the *FORM Reference* manual,
- the SCRLAY and SCRIMG classes described in the *FORM Reference* manual,
- use of the Series 3 Word application.

Class diagram



The document class DOC may be (a subclass of) either of the FORM classes EPFDOC or EPDOC, or it may be an application-specific class that is compatible with these classes. See the *Formatted Document Classes* chapter of the *FORM Reference* manual for further details.

EDWIN

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ wn_visible lg_draw lg_self_check ~~lg_set_id_pos~~ ~~wn_init~~ ~~lg_sense_width~~ *lg_update*

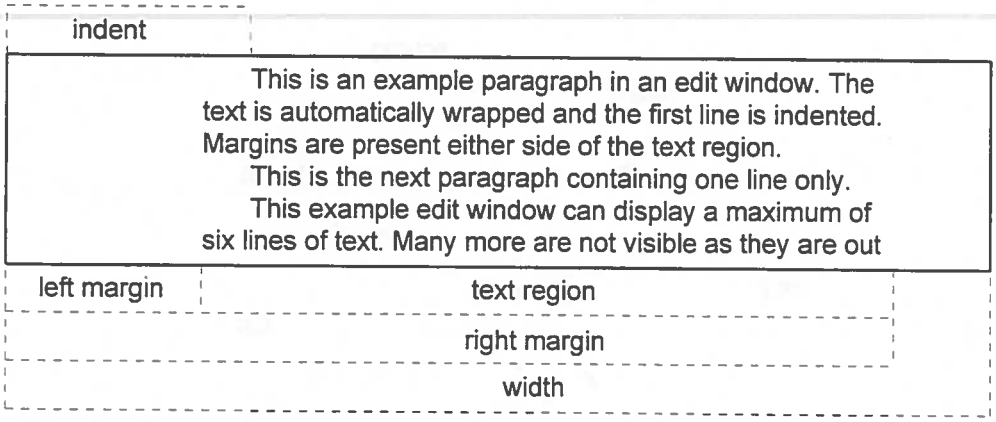
EDWIN	
scrimg	select
scrlay	change
doc	flags
cpos	margins
clen	font
clip	
destroy	ew_snuggle_insert
wn_init	ew_set_font
wn_key	ew_leave
wn_draw	ew_find
wn_sense_help	ew_replace
wn_set	ew_evaluate
wn_sense	ew_replace_clip
wn_emphasise	ew_paste_clip
lg_set_id_pos	ew_bring_in
lg_sense_width	ew_ep_insert
ew_sense_size	ew_return_key
ew_set_size	ew_tab_key
ew_set	ew_init_style
ew_sense	ew_readonly
ew_insert	

The EDWIN class provides the functionality of a window containing a formatted display of editable text. It is highly flexible since, in addition to being suitable for use as a simple one-line edit box in a dialog, it also supplies most of the basic functionality of a word processor. A subclass of EDWIN is, for example, used by the Word application.

Note that a tutorial introduction to the EDWIN class may be found in the *Edit Windows* chapter of the *Object Oriented Programming Guide*.

The edit window

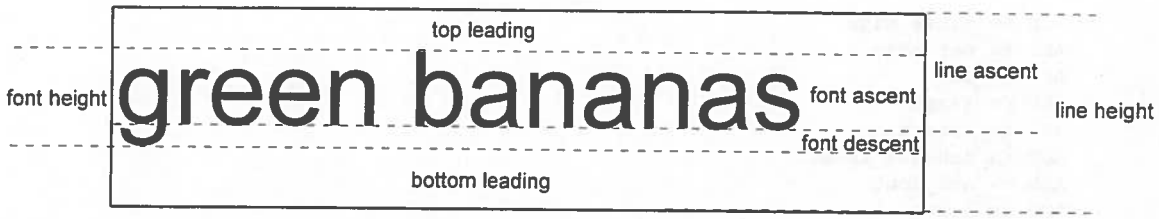
In general, the text in an edit window is arranged as a sequence of paragraphs. The text of each paragraph is automatically word wrapped within the window, as illustrated schematically in the following diagram:



The margins can be altered by setting the `edwin.margins` property, and the size and position of the window can be altered by means of the `ew_set_size` method.

Text lines

The text lines in an edit window consist of text spaced out with top and bottom leading as illustrated in the following picture:



(Note: leading derives from the metal used by typesetters.)

The font ascent, font descent and font height are characteristics of the font and the font style: see the *Window Server* manual for details of fonts.

The font and style are by default global to the document: defining local styles and fonts is described in the *Object Oriented Programming* manual.

The top leading is the space at the top of the line whilst the bottom leading is the space at the bottom of the line. The space between two lines is thus the sum of the topa and bottom leading - i.e. the total leading. The leading is global .

The font, the style, and the leading can be altered using the `ew_set_font` method.

Cursor position

The position of the text cursor is simply the index of the preceding character - thus a cursor placed at the start of the document has a position of zero whilst a cursor placed immediately after the one hundredth character in the document has a cursor position of one hundred.

Line cursor

The SCRIMG document imaging class divides the drawing area into three columns consisting of a labels margin, a line cursor margin and a text region: by default the width of the first two columns is set to zero in the `wn_init` method. However a line cursor, which indicates the current line, may be specified on initialisation. The effect is illustrated by, for example, the Word application's main edit window.

Link paste

An edit window can act as a link-paste client using the `bring_in` method. It must however be subclassed to allow it to act as a link-paste server. For details of the link-paste mechanism see the *Link Paste* chapter of the *Object Oriented Programming* manual.

Class definition

Defined in sub-category file *edwin.cl* (generated header file *edwin.g*).

```
CLASS   edwin   lodger
Editable text window
    {
    REPLACE destroy
    REPLACE wn_init
    REPLACE wn_key
    REPLACE wn_draw
    REPLACE wn_sense_help
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_emphasise
    REPLACE lg_set_id_pos
    REPLACE lg_sense_width
```

```
ADD ew_sense_size
ADD ew_set_size
ADD ew_set
ADD ew_sense
ADD ew_insert
ADD ew_snuggle_insert
ADD ew_set_font
ADD ew_leave
ADD ew_find
ADD ew_replace
ADD ew_evaluate
ADD ew_replace_clip
ADD ew_paste_clip
ADD ew_bring_in
ADD ew_ep_insert
ADD ew_return_key=p_false
ADD ew_tab_key=p_false
ADD ew_init_style=p_dummy
ADD ew_readonly=p_true

CONSTANTS
    {
    PR_EDWIN_DIALLABLE              0x0001
    PR_EDWIN_ACCEPT_TABS            0x0002
    PR_EDWIN_ACCEPT_SOFT_HYPHENS    0x0004
    PR_EDWIN_AUTO_CUR_END           0x0008
    PR_EDWIN_AUTO_SELECT            0x0010
    PR_EDWIN_DOC_SUPPLIED           0x0020
    PR_EDWIN_CLIPBOARD              0x0040
    PR_EDWIN_NOTIFY_OVERFLOW        0x0080  Same as IN_EDWIN_VULEN
    PR_EDWIN_READONLY               0x8000

    IN_EDWIN_DIALLABLE              PR_EDWIN_DIALLABLE
    IN_EDWIN_ACCEPT_TABS            PR_EDWIN_ACCEPT_TABS
    IN_EDWIN_ACCEPT_SOFT_HYPHENS    PR_EDWIN_ACCEPT_SOFT_HYPHENS
    IN_EDWIN_AUTO_CUR_END           (PR_EDWIN_AUTO_CUR_END|PR_EDWIN_AUTO_SELECT)
    IN_EDWIN_NO_AUTOSELECT          PR_EDWIN_AUTO_SELECT
    IN_EDWIN_DOC_SUPPLIED           PR_EDWIN_DOC_SUPPLIED
    IN_EDWIN_CLIPBOARD              PR_EDWIN_CLIPBOARD
    IN_EDWIN_VULEN                  0x0080
    IN_EDWIN_VULEN_NOSCALE          0x0100
    IN_EDWIN_VULEN_CHARACTERS       IN_EDWIN_VULEN
    IN_EDWIN_VULEN_PIXELS           (IN_EDWIN_VULEN|IN_EDWIN_VULEN_NOSCALE)
    IN_EDWIN_LEFT_CURSOR            0x0200
    IN_EDWIN_TEXT_SEGMENTED         0x0400
    IN_EDWIN_POSITION_SUPPLIED      0x0800
    IN_EDWIN_FONT_SUPPLIED          0x1000
    IN_EDWIN_LEADING_SUPPLIED       0x2000
    IN_EDWIN_VISLINES_SUPPLIED      0x4000
    IN_EDWIN_PAGINATABLE            0x8000

    SET_EDWIN_TXT                   0x01
    SET_EDWIN_EMPTY                 0x02
    SET_EDWIN_CUR_END               0x04
    SET_EDWIN_SEL_ALL               0x08
    SET_EDWIN_CURSOR                0x10
    SET_EDWIN_ANCHOR                0x20

    EWF_BACKWARDS   0x0001    direction to scan (0=forwards,1=back)
    EWF_CASESENS    0x0002    TRUE iff case-sensitive search
    EW_CHANGE_SINCE_SAVED           0x01
    EW_CHANGE_SINCE_PAGINATE        0x02
    EW_CHANGE                       0xffff
    EW_BRING_SINGLE_SHOT            0xf000
    }
```

```
TYPES
    {
    typedef struct
        {
        UWORD vulen;          viewing length or width
        UWORD flags;          autoselect etc
        UWORD maxlen;         maximum number of characters allowed
        TEXT contents[1];     rest of initial contents follows in line
        } IN_EDWIN;
    typedef struct
        {
        WORD total;
        WORD top;
        } EDWIN_LEADING;

    typedef struct
        {
        UWORD vislines;              number of lines visible in window
        P_POINT pos;                 top left offset relative to landlord
        WORD font;                   font ID
        UWORD style;                 font style
        EDWIN_LEADING leading;       total and top-only vertical leadings
        VOID *doc;                   document object to use
        VOID *clip;                  possible clipboard to use
        } IN_EDWIN_X;   never used in edwins in dialogs

    typedef struct
        {
        TEXT *buf;
        UWORD len;
        } SE_EDWIN;
    typedef struct
        {
        UWORD flags;
        SE_EDWIN txt;
        UWORD cursor;  cursor (moving point of select)
        UWORD anchor;  anchor point (fixed end of select)
        } SET_EDWIN;

    typedef struct
        {
        UWORD cursor;  cursor (moving point of select)
        UWORD anchor;  anchor point (may equal cursor)
        } SENSE_EDWIN;
    }

PROPERTY 2
    {
    VOID *scrimg;           screen imager
    VOID *scrlay;           screen layout
    VOID *doc;              document content
    UWORD cpos;             current position
    UWORD clen;             total content
    VOID *clip;             clipboard
    UWORD select;           TRUE if there is a select region
    UWORD change;           holds EW_CHANGE_ flags
    UWORD flags;            holds PR_EDWIN_ flags
    SCRLAY_MARGINS margins;
    SCRLAY_FONT font;
    }
}
```

Property

`edwin.scrimg` — the handle of an instance of SCRIMG: responsible for the screen display. See the *Document Layout Classes* chapter of the *FORM Reference* manual for details.

`edwin.scrlay` — the handle of an instance of SCRLAY: responsible for document layout. See the *Document Layout Classes* chapter of the *FORM Reference* manual for details.

`edwin.doc` — the handle of an instance of EPDOC or EPFDOC: contains the document text. See the *Editable Documents* chapter of the *FORM Reference* manual for details.

edwin.cpos	the current cursor position
edwin.clen	total length of the document ignoring the terminating zero
edwin.clip	the handle of an instance of EPFLAT or EPSEG: contains the clipboard text
edwin.select	TRUE if a region of text is selected, FALSE otherwise
edwin.change	set to indicate that the document has been edited: it is the responsibility of the application to reset this to FALSE once the changes have been recorded.
edwin.flags	an ORed combination of flags which determines the behaviour and content of the text window: the flags are described in the description of the methods
edwin.margins	the global settings for the margins
edwin.font	the global font information

(Note that doc and clip are not on the auto-destroy list of EDWIN - unlike scrimg and scrlay - as they can be created and looked after by the owner of the EDWIN instance and may thus survive the destruction of the EDWIN instance.)

EDWIN methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

If PR_EDWIN_DOC_SUPPLIED is not set in edwin.flags, and edwin.doc is non-zero, sends a DESTROY message to edwin.doc.

If PR_EDWIN_CLIPBOARD is set in edwin.flags, and edwin.clip is non-zero, sends a DESTROY message to edwin.doc.

Supersends a DESTROY message.

WN_INIT — Create components and initialise

```
VOID wn_init(IN_EDWIN *init, PR_WIN *landlord, IN_EDWIN_X *initx);
```

Initialise the edit window: landlord should specify the window ID of the landlord window.

The initial content and behaviour of the edit window are specified by means of an IN_EDWIN struct and an IN_EDWIN_X struct.

The IN_EDWIN struct is defined as follows:

```
typedef struct
    {
    UWORD vulen;
    UWORD flags;
    UWORD maxlen;
    TEXT contents[1];
    } IN_EDWIN;
```

The members of the IN_EDWIN struct have the following significance:

vulen	the width of the view in characters or pixels.
flags	an ORed combination of flags: see below.
maxlen	the maximum number of characters in the document: excludes the final zero terminator.
contents	the initial content of the document.

The IN_EDWIN_X struct is defined as follows:

```
typedef struct
    {
    UWORD vislines;
    P_POINT pos;
    WORD font;
    UWORD style;
    EDWIN_LEADING leading;
    VOID *doc;
    VOID *clip;
    } IN_EDWIN_X;
```

The members of the IN_EDWIN_X struct have the following significance:

vislines	the maximum number of lines of text visible in the edit window.
pos	the top left offset of the edit window relative to the landlord window.
font	the global font ID
style	the global font style.
leading	the vertical line spacing.
doc	NULL or the handle of an instance of either EPDOC or EPDOC: contains the document text.
clip	NULL or the handle of an instance of EPFLAT: contains the clipboard text.

The initialisation is controlled by ORing a suitable combination of the following flags into the flags member of the IN_EDWIN struct:

IN_EDWIN_NO_AUTOSELECT	Do not create a selected region: by default the entire document is selected.
IN_EDWIN_FONT_SUPPLIED	Use the font and style specified in initx->font and initx->style respectively: the default values are WS_FONT_SYSTEM and G_STY_NORMAL respectively.
IN_EDWIN_VULEN_CHARACTERS	Set the width of the edit window as specified in init->vulen: units of characters.
IN_EDWIN_VULEN_PIXELS	Set the width of the edit window as specified by init->vulen: units of pixels.
IN_EDWIN_PAGINATABLE	For advanced use only.
IN_EDWIN_TEXT_SEGMENTED	If this flag is set, any document object created during initialisation will be an instance of EPDOC, which uses segmented storage. In the absence of this flag (and assuming that the flag IN_EDWIN_DOC_SUPPLIED is also not set) an instance of EPFDOC.is created. Note that this flag *must* be set if the document object uses segmented storage (which is expected to be an instance of EPDOC or of a subclass of EPDOC) regardless of whether this instance is or is not created by the wn_init method.
IN_EDWIN_DOC_SUPPLIED	Use the document object specified in initx->doc. The default, in the absence of this flag, is to create the document object.
IN_EDWIN_CLIPBOARD	Use the clipboard object specified in initx->clip , or, if initx->clip is zero create a clipboard object with maximum length as specified in init->maxlen. The default is not to use a clipboard.
IN_EDWIN_POSITION_SUPPLIED	Use the top left offset of the edit window specified by initx->pos. If this flag is not set the offset must be set by a subsequent call to lg_set_id_pos.

IN_EDWIN_VISLINES_SUPPLIED	Use the number of text lines visible specified in initx->vislines. The default is to show only one text line - i.e. a single text line editor.
IN_EDWIN_LEADING_SUPPLIED	Use the leading specified in initx->leading.top and initx->leading.total. The default is to use top and total leading of one and two pixels respectively.
IN_EDWIN_ACCEPT_TABS	Accept Tab key presses. The default is to ignore Tab key presses.
IN_EDWIN_ACCEPT_SOFT_HYPHENS	Allow soft hyphens. The default is to ignore soft hyphens.
IN_EDWIN_LEFT_CURSOR	Draw a line cursor in the left margin.

The above information is quite sufficient for simple use of the wn_init method. The following detailed explanation of the operation of the method is included for more advanced use.

Sets the landlord window ID into property by writing landlord to lodger.landlord.

Writes init->flags to edwin.flags:

- allows read and write access to the document by clearing PR_EDWIN_READONLY in edwin.flags.
- if IN_EDWIN_NO_AUTOSELECT is set (clear) in init->flags clears (sets) PR_EDWIN_AUTOSELECT in edwin.flags.

Sets the font and font style:

- if init->flags contains IN_EDWIN_FONT_SUPPLIED, writes initx->font to edwin.font.fid and writes initx->style to edwin.font.style: defines user specified font and font style.
- otherwise uses the system font, by writing W_FONT_SYSTEM to edwin.font.fid.

In the case of the Work*about*, if the edit window is being used as a control in a small font dialog, edwin.font.fid is set to the font indicated by the value of dlgbox.font in the owning dialog box.

Sets the width of the lodger window:

- if init->flags contains neither IN_EDWIN_VULEN_PIXELS nor IN_VULEN_CHARACTERS, writes init->maxlen multiplied by the maximum width of a character in the current font to lodger.width.
- if init->flags contains IN_EDWIN_VULEN_PIXELS writes init->vulen to lodger.width.
- if init->flags contains IN_EDWIN_VULEN_CHARACTERS writes init->vulen multiplied by the nominal maximum width of a character in the current font to lodger.width.

If init->flags contains IN_EDWIN_DOC_SUPPLIED:

- writes initx->doc to edwin.doc. This is assumed to be the handle of (a subclass of) either EPDOC or EPFDOC, depending on the presence or absence of the IN_EDWIN_TEXT_SEGMENTED flag.

Otherwise creates and initialises a document object:

- if init->flags contains IN_EDWIN_TEXT_SEGMENTED creates an instance of EPDOC and writes the handle to edwin.doc.
- otherwise creates an instance of EPFDOC and writes the handle to edwin.doc.
- sends an EP_INIT message to edwin.doc specifying a maximum length of init->maxlen characters plus a zero terminator.
- copies the text specified by init->content into the document by sending an EP_SET_TEXT messsage to edwin.doc.

Senses the document length by sending an EP_SENSE_LEN message to edwin.doc and writes the result to edwin.clen.

Builds a SCRLAY_DOC struct containing document information and methods required by the SCRLAY component. If init->flags contains IN_EDWIN_TEXT_SEGMENTED (on the assumption that the document content object is, or is a subclass of, EPDOC):

- writes O_EPDOC_SENSE_CHARS to the sensechars member.
- writes O_EPDOC_PARA_START to the toparst member.
- if init->flags contains IN_EDWIN_PAGINATABLE, writes O_EPDOC_ENQ_PAGE to the enqpage member.

Otherwise (on the assumption that the document content object is, or is a subclass of, EPFDOC) sets the above members as follows:

- writes O_EPFDOC_SENSE_CHARS to the sensechars member.
- writes O_EPFDOC_PARA_START to the toparst member.

Sets the remaining members as follows:

- writes zero to the sensepdata and senseplabel members: these are thus undefined.
- writes edwin.doc to the content member: defines the document object.
- sends an EP_SENSE_LEN message to edwin.doc and writes the return value plus one to the len member: defines the document length.

Builds a SCRLAY_STYLE struct containing style information required by the SCRLAY component as follows:

- writes SCRLAY_SHOW_TABS to the options member: defines tabs as visible.
- writes FALSE to the printer member: defines the layout mode as *screen*.
- writes a pointer to edwin.margins to the pd.margins member: defines the margins.
- writes the address of edwin.font to the font and sfont members: defines the global printer and screen fonts.
- writes the address of edwin.font.height to the pd.tabs member: defines zero tabs.
- writes NULL to the fwtab member. the global printer font width table is undefined.
- writes zero to the scrpwidth member: the printer tab positions information is undefined.

Allows subclassers to change the default style settings in the SCRLAY_STYLE struct by sending self an EW_INIT_STYLE message. (The default method does nothing.).

Creates an instance of SCRLAY and writes the handle to edwin.scrlay. Initialises by sending an SL_INIT message to edwin.scrlay with pointers to the above SCRLAY_DOC and SCRLAY_STYLE structs.

If init->flags contains IN_EDWIN_CLIPBOARD and initx->clip is zero, creates an instance of EPFLAT and writes the handle to edwin.clip. Initialises with maximum length init->maxlen characters excluding the zero terminator by sending an EP_INIT message to edwin.clip.

If init->flags contains IN_EDWIN_CLIPBOARD, and initx->clip is non-zero, writes initx->clip to edwin.clip and clears IN_EDWIN_CLIPBOARD from edwin.flags as the clipboard is now initialised.

Builds an SCRIMG_WIN struct containing window information. If init->flags contains IN_EDWIN_LEFT_CURSOR:

- writes W_FONT_SYSTEM to win.lcfont: defines the line cursor font.
- if G_STY_DOUBLE is set in edwin.font.style, sets G_STY_DOUBLE in win.lcstyle: defines the line cursor style - zero for the default style.
- writes WS_SYMBOL_MARGIN_CURSOR to win.lccode: defines the line cursor symbol.

If init->flags contains IN_EDWIN_LEADING_SUPPLIED:

- writes the sum of the font height in pixels and initx->leading.total to win.lheight: defines the line height.
- writes the sum of the font ascent in pixels and initx->leading.top to win.lascent: defines the line ascent.

Otherwise sets the above members as follows:

- writes the font height in pixels plus two to `win.lheight`: defines the line height.
- writes the font ascent in pixels plus one to `win.lascent`: defines the line ascent.

If `init->flags` contains `IN_EDWIN_VISLINES`:

- writes `initx->vislines` to `win.nlines`: defines the maximum number of lines visible.
- writes `win.width` divided by four to `win.hscrlm`: defines the horizontal scroll behaviour.

Otherwise:

- writes one to `win.nlines`: defines the maximum number of lines visible.
- writes zero to `win.hscrlx` and writes two to `win.hscrlm`: defines the horizontal scroll behaviour.

Sets the remaining members as follows:

- writes `landlord->win.id` to `win.wid`: defines the landlord window ID.
- writes `initx->pos` to `win.tl`: defines the top left offset of the window.
- writes `lodger.width` to `win.width`: defines the width of the edit window.
- writes zero to `win.margin`: defines the default width of the label and line cursor margins.
- writes two to `win.cwidth`: defines the width of the text cursor.

Creates an instance of `SCRIMG` and write the handle to `edwin.scrimg`.

Sets the window information by sending an `SI_SET` message to `edwin.scrimg` with as arguments a pointer to the above `SCRIMG_WIN` struct and `edwin.scrlay`.

Subtracts the font maximum character width from the pixel width of the text area of the edit window (this is the return value from the last `SI_SET` message) and writes the result to `edwin.margins.right`.

If `win.nlines` is one in which case only one line is visible:

- disables the word wrap by writing 4096 to `edwin.margins.right`.

If `init->flags` contains `IN_EDWIN_POSITION_SUPPLIED`:

- sets the offset of the lodger window by writing `win.tl` to `lodger.offset`.
- sets the lodger window ID by writing `win.wid` to `win.id`.
- sends an `SI_INIT` message to `edwin.scrimg` with as arguments zero and zero.

(Otherwise does nothing and expects the caller to send an `LG_SET_ID_POS` message later.)

If `edwin.flags` contains `IN_EDWIN_POSITION_SUPPLIED` and either `IN_EDWIN_AUTO_CUR_END` or `IN_EDWIN_AUTO_SELECT`:

- moves the cursor to the end of the document and scrolls the view until the cursor is visible by sending an `SI_MOVE_CURSOR` message to `edwin.scrimg`.
- writes the length of the document excluding the terminating zero to `edwin.cpos`.
- if `edwin.flags` contains `IN_EDWIN_AUTO_SELECT` selects the region defined by the old and new cursor positions.

WN_KEY — Handle key input

```
INT wn_key(UINT keycode, UINT modifiers);
```

Handle a keypress.

If the keypress is `W_KEY_ESCAPE`:

- cancels any selection, write `FALSE` to `edwin.select` and returns `WN_KEY_NO_CHANGE`.

If the keypress is W_KEY_DELETE_RIGHT:

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.
- if a selected region exists, moves the cursor to the start of the selection, delete the selection, and scrolls the view until the cursor is visible. Writes the cursor position to edwin.cpos. Writes FALSE to edwin.select. Writes the document length to edwin.clen. Rebuilds the layout and the view - the method sends an SI_FWD_CHANGED message to edwin.scrimg. Writes EW_CHANGE to edwin.change. Returns WN_KEY_CHANGED.
- if the end of the document has been reached, returns WN_KEY_NO_CHANGE.
- otherwise deletes the character to the right of the cursor - the method reformats and redraws the edited document responsively by sending an SI_PARA_CHANGED message to edwin.doc. Writes the document length to edwin.clen. Writes EW_CHANGE to edwin.change. Return WN_KEY_CHANGED.

If the keypress is W_KEY_DELETE_LEFT:

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.
- if modifiers contains W_PSION_MODIFIER, clear PR_WSERV_INSERT_MODE and PR_WSERV_INSERT_PENDING in w_ws->wserv.flags. Selects the region to the left of the cursor and moves the cursor to the start of the line by sending self a WN_KEY message with arguments of W_KEY_HOME and W_SHIFT_MODIFIER.
- if a region is selected, moves the cursor to the start of the selection, deletes the selection, and scrolls the view until the cursor is visible. The selection is copied to the clipboard object if one exists. Writes the cursor position to edwin.cpos. Writes FALSE to edwin.select. Writes the document length to edwin.clen. Rebuilds the layout and the view - the method sends an SI_FWD_CHANGED message to edwin.scrimg. Writes EW_CHANGE to edwin.changed and returns WN_KEY_CHANGED.
- if the cursor is at the start of the document returns WN_KEY_NO_CHANGE.
- otherwise deletes the character to the left of the cursor, moves the cursor to the left by one character and scrolls the view until the cursor is visible: the method sends an SI_DELPREP message to edwin.scrimg, an EP_DELETE message to edwin.doc and an SI_PARA_CHANGED changed message to edwin.scrimg. Writes the length of the document to edwin.clen. Writes the cursor position to edwin.cpos. Writes EW_CHANGE to edwin.changed and returns WN_KEY_CHANGED.

If the keypress is W_KEY_LEFT:

- if a selected region exists and modifiers does not contain W_SHIFT_MODIFIER, moves the cursor to the start of the selected region, cancels the selection and scrolls the view until the cursor is visible. Writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.
- if the cursor is at the start of the document returns WN_KEY_NO_CHANGE.
- if modifiers contains W_CTRL_MODIFIER, moves the cursor backwards to the start of a word - the method sends an EP_SCAN_WORD message to edwin.doc - and scrolls the view until the cursor is visible. Writes the cursor position to edwin.cpos. If modifiers contains W_SHIFT_MODIFIER selects the text skipped and writes TRUE to edwin.select. Returns WN_KEY_NO_CHANGE.
- otherwise moves the cursor backwards one character and scrolls the view until the line containing the cursor is visible. Writes the cursor position to edwin.cpos. If modifiers contains W_SHIFT_MODIFIER selects the character skipped and writes TRUE to edwin.select. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_RIGHT:

- if a selected region exists and modifiers does not contain W_SHIFT_MODIFIER, moves the cursor to the end of the selected region, cancels the selection and scrolls the view until the cursor is visible. Writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.
- if the cursor is at the end of the document, returns WN_KEY_NO_CHANGE.
- if modifiers contains W_CTRL_MODIFIER, moves the cursor forwards to the start of a word - the method sends an EP_SCAN_WORD message to edwin.doc and an SI_MOVE_CURSOR message to

edwin.scrimg - and scrolls the view until the cursor is visible. Writes the cursor position to edwin.cpos. If modifiers contains W_SHIFT_MODIFIER selects the text skipped and writes TRUE to edwin.select. Otherwise writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

- otherwise moves the cursor forwards by one character and scrolls the view until the cursor is visible. Writes the cursor position to edwin.cpos. If modifiers contains W_SHIFT_MODIFIER selects the character skipped and writes TRUE to edwin.select. Otherwise writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_HOME:

- if modifiers contains both W_CTRL_MODIFIER and W_SHIFT_MODIFIER, moves the cursor backwards until a word delimiter is located and then selects the word after the delimiter - i.e. selects the current word. The method sends an EP_SCAN_WORD message to edwin.doc and an SI_MOVE_CURSOR message to edwin.scrimg.
- otherwise moves the cursor to the beginning of the current line. If modifiers contains W_SHIFT_MODIFIER selects the text skipped and writes TRUE to edwin.select. Otherwise cancels any selected region and writes FALSE to edwin.select.

If the keypress is W_KEY_END:

- if modifiers contains both W_CTRL_MODIFIER and W_SHIFT_MODIFIER, moves the cursor forwards until a paragraph delimiter is located then selects the paragraph preceding the cursor - i.e. selects the current paragraph. The method sends an EP_SCAN_PARA message to edwin.doc and an SI_MOVE_CURSOR message to edwin.scrimg.
- otherwise moves the cursor to the end of the current line. If modifiers contains W_SHIFT_MODIFIER selects the text skipped and writes TRUE to edwin.select. Otherwise cancels any selection and writes FALSE to edwin.select.

If the keypress is W_KEY_UP:

- if a region is selected and modifiers does not contain W_SHIFT_MODIFIER, cancels the selection and writes FALSE to edwin.select. Moves the cursor to the start of the once selected region and scrolls the view until the cursor is visible. Returns WN_KEY_NO_CHANGE.
- if modifiers contains W_CTRL_MODIFIER, moves the cursor to the start of the current paragraph and scrolls the view until the cursor is visible. If modifiers also contains W_SHIFT_MODIFIER selects the characters skipped and writes TRUE to edwin.select. Otherwise cancels any selected region and writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.
- otherwise moves the cursor up one line and scrolls the view until the cursor is visible. If modifiers contains W_SHIFT_MODIFIER, and a region is not selected, selects the text skipped and writes TRUE to edwin.select. If modifiers contains W_SHIFT_MODIFIER, and a region is selected, moves the cursor point of the selected region to the new cursor position. Otherwise writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_DOWN:

- if a region is selected and modifiers does not contain W_SHIFT_MODIFIER, moves the cursor to the end of the selected region, cancels the selection, and scrolls the view until the cursor is visible. Writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.
- if modifiers contains W_CTRL_MODIFIER, moves the cursor to the start of the first line of the next paragraph and scrolls the view until the cursor is visible. If modifiers also contains W_SHIFT_MODIFIER selects the characters skipped and writes TRUE to edwin.select. Returns WN_KEY_NO_CHANGE.
- otherwise moves the cursor down one line and scrolls the view until the cursor is visible. If modifiers contains W_SHIFT_MODIFIER, and a region is not selected, selects the text skipped and writes TRUE to edwin.select. If modifiers contains W_SHIFT_MODIFIER, and a region is selected, moves the cursor point of the selected region to the new cursor position. Otherwise cancels any selection and writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_PAGE_UP:

- if modifiers contains W_CTRL_MODIFIER, moves the cursor to the start of the document and writes the cursor position to edwin.cpos. If modifiers also contains W_SHIFT_MODIFIER selects the text

skipped and writes TRUE to edwin.select. Otherwise cancels any selection and writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

- otherwise moves the cursor up by one 'page' (that is, by one fewer lines than the number of lines that can be displayed on the screen) and writes the cursor position to edwin.cpos. If modifiers also contains W_SHIFT_MODIFIER selects the text skipped and writes TRUE to edwin.select. Otherwise cancels any selection and writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_PAGE_DOWN:

- if modifiers contains W_CTRL_MODIFIER, moves the cursor to the end of the document and scrolls the view until the cursor is visible. Writes the cursor position to edwin.cpos. If modifiers contains W_SHIFT_MODIFIER, selects the text skipped and writes TRUE to edwin.select. Otherwise cancels any selection and writes FALSE to edwin.select. Returns WN_KEY_NO_CHANGE.

- otherwise moves the cursor down by one 'page' (that is, by one fewer lines than the number of lines that can be displayed on the screen) and scrolls the view until the cursor is visible. Writes the cursor position to edwin.cpos. If modifiers contains W_SHIFT_MODIFIER, selects the text skipped and writes TRUE to edwin.select. Otherwise cancels any selection and write FALSE to edwin.select.

If the keypress is W_KEY_RETURN:

- checks that the document accepts W_KEY_RETURN by sending self an EW_RETURN_KEY message. If the return value is non-zero the method terminates, returning this value.

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.

- if modifiers contains W_SHIFT_MODIFIER, sets keycode to a carriage return - i.e. '\n'.

- otherwise sets keycode to zero - i.e. 0x00.

- inserts keycode into the document at the current cursor position and moves the cursor forwards one character. Writes the document length to edwin.clen. Cancels any selection and writes FALSE to edwin.select. Sends an SI_PARA_CHANGED message to edwin.scrimg. Writes EW_CHANGE to edwin.change. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_TAB:

- checks that the document accepts W_KEY_TAB by sending self an EW_TAB_KEY message. If the return value is non-zero returns the return value.

- if edwin.flags does not contain PR_EDWIN_ACCEPT_TABS, returns WN_KEY_NO_CHANGE.

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.

- inserts a tab character at the current cursor position, moves the cursor to the right by one character and scrolls the view until the cursor is visible. Writes the document length to edwin.clen. Cancels any selection and writes FALSE to edwin.select. Sends an SI_PARA_CHANGED message to edwin.scrimg. Returns WN_KEY_NO_CHANGE.

If the keypress is W_KEY_HELP:

- if modifiers does not contain *both* W_PSION_MODIFIER *and* W_SHIFT_MODIFIER, *and* edwin.flags does not contain PR_EDWIN_DIALLABLE, returns WN_KEY_NO_CHANGE.

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.

- if modifiers does not contain W_CTRL_MODIFIER, cancels then deletes any selected region - the selected region is copied to the clipboard. Inserts a WS_PHONE_SYMBOL character at the current cursor position, moves the cursor forwards one character and scrolls the view until the cursor is visible. Writes the document length to edwin.clen. Writes the cursor position to edwin.cpos. Writes FALSE to edwin.select. Writes EW_CHANGE to edwin.change and sends an SI_FWD_CHANGE message to edwin.scrimg. Returns WN_KEY_CHANGED.

- otherwise runs a country selector dialog by sending a WS_APPEND_COUNTRY message to w_ws and inserts the selected country name into the document by sending self an EW_INSERT message. Returns WN_KEY_NO_CHANGE.

If the keypress is ' ':

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.
- if a selected region exists, copies the selected region to the clipboard and then cancels and deletes the selected region. Writes the document length to edwin.clen. Writes FALSE to edwin.select.
- if modifiers contains W_CTRL_MODIFIER, replaces the value of keycode by WS_SYMBOL_HARD_SPACE.
- inserts keycode.
- moves the cursor forwards one character and, if necessary, scrolls the view until the cursor is visible. Writes the document length to edwin.clen. Writes the cursor position to edwin.cpos.
- if a select region has been deleted, sends an SI_DOC_CHANGED message to edwin.scrimg, otherwise sends edwin.scrimg an SI_PARA_CHANGED message. Writes EW_CHANGE to edwin.change. Returns WN_KEY_CHANGED.

If the keypress is '-':

- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.
- if a selected region exists copies the selected region to the clipboard and then cancels and deletes the selected region. Writes the document length to edwin.clen. Writes FALSE to edwin.select.
- if modifiers contains W_SHIFT_MODIFIER and W_CTRL_MODIFIER, the value of keycode is replaced by WS_SYMBOL_HARD_HYPHEN.
- otherwise, if modifiers contains W_CTRL_MODIFIER and edwin.flags contains PR_EDWIN_ACCEPT_SOFT_HYPHENS, the value of keycode is replaced by WS_SYMBOL_SOFT_HYPHEN.
- inserts keycode.
- moves the cursor forwards one character and, if necessary, scrolls the view until the cursor is visible. Writes the document length to edwin.clen and the cursor position to edwin.cpos.
- if a select region has been deleted, sends an SI_DOC_CHANGED message to edwin.scrimg, otherwise sends edwin.scrimg an SI_PARA_CHANGED message. Writes EW_CHANGE to edwin.change. Returns WN_KEY_CHANGED.

For any other keypress:

- if keycode does not specify a printable character or keycode is greater than or equal to 0x100, returns WN_KEY_NO_CHANGE.
- if edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.
- if a selected region exists, copies the selected region to the clipboard and moves to the start of the selected region scrolling the view until the cursor is visible. Cancels and deletes the selected region. Writes the document length to edwin.clen. Writes FALSE to edwin.select.
- inserts keycode at the current cursor position and moves the cursor forwards one character. Writes the document length to edwin.clen. Writes FALSE to edwin.select.
- if a selected region existed, rebuilds and redraws by sending an SI_DOC_CHANGED message to edwin.scrimg.
- otherwise rebuilds and redraws responsively by sending an SI_PARA_CHANGED message to edwin.scrimg.
- writes EW_CHANGE to edwin.change. Returns WN_KEY_CHANGED.

WN_DRAW — Draw view

```
VOID wn_draw(VOID);
```

Draw the whole view.

Sends an SI_REDRAW message to edwin.scrimg with an argument of NULL.

WN_SENSE_HELP — Sense help ID

```
INT wn_sense_help(VOID);
```

Sense the ID for EDWIN help resource.

Returns -SYS_HELP_EDIT.

WN_SET — Set text

```
VOID wn_set(SE_EDWIN *pset);
```

Set the content of the edit window according to the content of the SE_EDWIN struct pointed to by pset.

The SE_EDWIN struct is defined as follows:

```
typedef struct
    {
    TEXT *buf;
    UWORD len;
    } SE_EDWIN;
```

The members of the SE_EDWIN struct have the following significance:

buf — a pointer to the text to be set in the edit window

len — the total length of the text excluding the terminating zero

Sets the text in the edit window by sending an EP_SET_TEXT message to edwin.doc.

Writes the length of the document including the terminating zero to edwin.clen.

Cancels any selected region, moves the cursor to the start of the document and scrolls the view until the cursor is visible by sending an SI_DOC_RESET message to edwin.scrimg. Writes the cursor position to edwin.cpos.

If PR_EDWIN_AUTO_CUR_END is set in edwin.flags, moves the cursor to the end of the document and scrols the view until the cursor is visible by sending an SI_MOVE_CURSOR message to edwin.scrimg . Writes the cursor position to edwin.cpos. Writes FALSE to edwin.select.

If PR_EDWIN_AUTO_SELECT is set in edwin.flags, moves the cursor position to the end of the document, scrolls the view until the cursor is visible and selects the whole document by sending an SI_MOVE_CURSOR message to edwin.scrimg. Writes the cursor position to edwin.cpos. Writes TRUE to edwin.select.

WN_SENSE — Sense text

```
VOID wn_sense(SE_EDWIN *psense);
```

Sense the text in the edit window and write the results to the SE_EDWIN struct pointed to by psense.

Senses the text by sending an EF_SENSE_BUF message to edwin.doc. Writes a pointer to the text to psense->buf and writes the length of the text minus the zero terminator to psense->len.

This method is not supported if the text is not stored contiguously.

WN_EMPHASISE — Set emphasis

```
VOID wn_emphasise(INT flag);
```

Emphasise the edit window.

Sends an SI_EMPHASIZE message to edwin.scrimg with an argument of flag.

LG_SET_ID_POS — Set window ID and position

```
VOID lg_set_id_pos(INT id, P_POINT *ppos, UINT width);
```

Set the offset of the lodger window as specified by ppos, the width of the lodger window as specified by width, and the ID of the lodger window as specified by id.

Sets the offset width, and ID of the lodger window by supersending an LG_SET_ID_POS message.

Obtains a copy of the SCRIMG window information data structure by sending an SI_SENSE message to edwin.scrimg with as argument a pointer to a SCRIMG_WIN struct.

Writes the struct pointed to by ppos to the tl member of the SCRIMG_WIN struct. Writes width to the width member of the SCRIMG_WIN struct.

Initialises the SCRIMG instance and sets the content of the above SCRIMG_WIN struct into property by sending an SI_INIT messsage to edwin.scrimg.

If PR_EDWIN_AUTO_CUR_END is set in edwin.flags, moves the cursor to the end of the document by sending an SI_MOVE_CURSOR message to edwin.scrimg. Writes the cursor position to edwin.cpos. Writes FALSE to edwin.select.

If PR_EDWIN_AUTO_SELECT is set in edwin.flags, moves the cursor to the end of the document and selects the whole document by sending an SI_MOVE_CURSOR message to edwin.scrimg. Writes the cursor position to edwin.cpos. Writes TRUE to edwin.select.

LG_SENSE_WIDTH — Sense width

```
INT lg_sense_width(VOID);
```

Sense the width of the lodger window.

Returns lodger.width.

EW_SENSE_SIZE — Sense screen image size

```
VOID ew_sense_size(P_EXTENT *pext);
```

Sense the top left offset of the lodger window, the width of the lodger window and the maximum number of text lines visible and write the results to the P_EXTENT struct specified by pext.

Senses the window data for the SCRIMG instance by sending an SI_SENSE message to edwin.scrimg.

Writes the pixel coordinates of the top left corner of the lodger window to pext->tl. Writes the pixel width of the lodger window to pext->width. Writes the number of text lines visible to pext->height.

EW_SET_SIZE — Set screen image size data

```
VOID ew_set_size(P_EXTENT *pext, VOID *hand, INT method);
```

Set the top left offset of the edit window as specified by pext->tl, the width of the lodger window as specified by pext->width and the maximum number of text lines visible as specified by pext->height.

Senses the current window data for the SCRIMG component by sending an SI_SENSE message to edwin.scrimg with as argument a pointer to an SCRIMG_WIN struct.

Writes the new offset of the lodger window pext->tl to lodger.offset.

Writes the new coordinates of the top left point of the edit window pext->tl to the tl member of the SCRIMG_WIN struct.

Writes pext->height, which contains the new number of text lines visible, to the `nlines` member of the `SCRIMG_WIN` struct.

Sets the modified window data into the `SCRIMG` component by sending an `SI_SET` message to `edwin.scrimg`.

If `hand` is non-zero, sends a `method` message to `hand` with as argument the pixel width of the text area: this allows for further processing before the document layout is rebuilt.

Rebuilds the document layout and the view by sending an `SI_STYLE_CHANGED` message to `edwin.scrimg`.

EW_SET — Set content and cursor

```
VOID ew_set(SET_EDWIN *pset);
```

Set the content and appearance of the edit window according to the content of the `SET_EDWIN` struct specified by `pset`.

The `SET_EDWIN` struct is defined as follows:

```
typedef struct
    {
    UWORD flags;
    SE_EDWIN txt;
    UWORD cursor;  cursor (moving point of select)
    UWORD anchor;  anchor point (fixed end of select)
    } SET_EDWIN;
```

The members of the `SET_EDWIN` struct have the following significance:

`flags` see below.

`txt` an `SE_EDWIN` struct specifying the text to set in the edit window: see the description of the `wn_set` method for details.

`cursor` the candidate cursor position.

`anchor` the candidate anchor position.

The setting is controlled by `ORing` into `flags` a suitable combination of the following flags:

`SET_EDWIN_EMPTY` empty the content of the edit window

`SET_EDWIN_TXT` set the content of the edit window

`SET_EDWIN_SEL_ALL` select the entire content of the edit window

`SET_EDWIN_CUR_END` move the cursor to the end of the document

`SET_EDWIN_ANCHOR` set the position of the anchor for the selected region

`SET_EDWIN_CURSOR` set the position of the cursor

If `pset->flags` contains `SET_EDWIN_EMPTY`:

- sets `SET_EDWIN_TXT` in `pset->flags` and set `pset->txt.len` to zero

If `pset->flags` contains `SET_EDWIN_TXT`:

- sets the text in the edit window by sending an `EP_SET_TEXT` message to `edwin.doc` and writes the length of the text as specified by `pset->txt.len` to `edwin.clen`
- cancels any selection, moves the cursor to the start of the document and scrolls the view until the cursor is visible by sending an `SI_DOC_RESET` message to `edwin.scrimg`. Writes the cursor position to `edwin.cpos`.

If `pset->flags` contains `SET_EDWIN_SEL_ALL`:

- sets both `SET_EDWIN_CUR_END` and `SET_EDWIN_ANCHOR` in `pset->flags` and writes zero to `pset->anchor`

If pset->flags contains SET_EDWIN_CUR_END:

- sets SET_EDWIN_CURSOR in pset->flags and writes edwin.clen-1 to pset->cursor

If pset->flags contains SET_EDWIN_ANCHOR:

- writes pset->anchor to edwin.cpos, cancels any selected region, moves the cursor to edwin.cpos and scrolls the view until the cursor is visible by sending an SI_MOVE_CURSOR message to edwin.scrimg.

If pset->flags contains SET_EDWIN_CURSOR and SET_EDWIN_ANCHOR:

- selects the region between edwin.cpos and pset->cursor, moves the cursor to pset->cursor and scrolls the view until the cursor is visible by sending an SI_MOVE_CURSOR message to edwin.scrimg. Writes TRUE to edwin.select. Writes the cursor position to edwin.cpos.

If pset->flags contains SET_EDWIN_CURSOR but not SET_EDWIN_ANCHOR:

- cancels any selection, moves the cursor to pset->cursor and scrolls the view until the cursor is visible by sending an SI_MOVE_CURSOR message to edwin.scrimg. Writes FALSE to edwin.select. Writes the cursor position to edwin.cpos.

Note that the order of the above tests is significant: for example a region may be selected by ORing both SET_EDWIN_ANCHOR and SET_EDWIN_CURSOR into pset->flags.

EW_SENSE — Sense cursor and select data

```
INT ew_sense(SENSE_EDWIN *psense);
```

Sense the anchor and cursor position of the selected region and write the results to the SENSE_EDWIN struct specified by psense.

The SENSE_EDWIN struct is defined as follows:

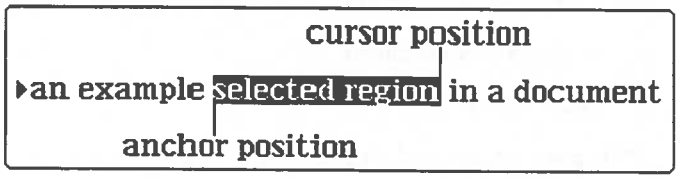
```
typedef struct
    {
    UWORD cursor;  cursor (moving point of select)
    UWORD anchor;  anchor point (may equal cursor)
    } SENSE_EDWIN;
```

The members of the SENSE_EDWIN struct have the following significance:

cursor — the cursor position of the selected region: equal to anchor plus the length of the selected region

anchor — the anchor position of the selected region

The anchor and cursor positions are illustrated in the following picture:



(Note that it is quite possible for the cursor position to come before the anchor position.)

Obtains the position of the first character and the length for the selected region by sending an SI_GET_SELECT message to edwin.scrimg and writes the anchor and cursor positions to psense.

Returns the number of characters in the selected region: zero indicates that no selected region exists.

EW_INSERT — Insert at cursor

```
VOID ew_insert(TEXT *buf, UINT blen);
```

Insert at the current cursor position `blen` characters from the buffer specified by `buf`.

Inserts the text into the document at the current cursor position by sending an `EP_INSERT` message to `edwin.doc`: the message is sent using `p_entersend` and on error an `EW_LEAVE` message is sent to `self` with the error code as argument.

Writes the length of the document to `edwin.clen`.

Moves the cursor to the end of the inserted text by sending an `SI_MOVE_CURSOR` message to `edwin.scrimg`: the view is scrolled until the cursor is visible. Writes the cursor position to `edwin.cpos`.

Writes `EW_CHANGE` to `edwin.change`.

Rebuilds the layout and view by sending an `SI_FWD_CHANGE` message to `edwin.scrimg`: the top left corner of the view is kept fixed.

EW_SNUGGLE_INSERT — Replace select and add surrounds

```
VOID ew_snuggle_insert(TEXT *before, TEXT *after, TEXT *replace);
```

Replace the selected region with the `before`, `replace` and `after` strings (in that order) and select the inserted `replace` text: `before`, `replace` and `after` may each specify a `NULL` address.

Checks that the sum of the current document length and the length of each insert string does not exceed the maximum allowed document length. If the maximum document length is exceeded, sends `self` an `EW_LEAVE` message with argument of `E_GEN_OVER`.

Obtains the length and position of the selected region by sending an `SI_GET_SELECT` message to `edwin.scrimg`.

If no selection exists, inserts at the current cursor position the `before`, `replace` and `after` strings by sending `EP_INSERT` messages to `edwin.doc`.

If a selection exists, inserts immediately after the selected region the `before`, `replace` and `after` strings by sending `EP_INSERT` messages to `edwin.doc`. Cancels the selection by sending an `SI_MOVE_CURSOR` message to `edwin.scrimg`, then deletes the once selected region by sending an `EP_DELETE` message to `edwin.doc`.

Writes `EW_CHANGE` to `edwin.change` and sends an `SI_DOC_CHANGED` message to `edwin.scrimg`.

Selects the inserted `replace` string, moves the cursor to the end of the selected region, and scrolls the view until the cursor is visible by sending `SI_MOVE_CURSOR` messages to `edwin.scrimg`.

EW_SET_FONT — Set font and line spacing

```
VOID ew_set_font(INT font,UINT style,EDWIN_LEADING *leading);
```

Set the font as specified by `font`, the font style as specified by `style`, and the line spacing as specified by `leading`.

Writes `font` to `edwin.font.fid` and writes `style` to `edwin.font.style`.

Senses the window information for the `SCRIMG` instance by sending an `SI_SENSE` message to `edwin.scrimg` with as argument a pointer to an `SCRIMG_WIN` struct.

Writes the line height in pixels to the `lheight` member of the `SCRIMG_WIN` struct: the line height is equal to the sum of `leading->total` and the font height.

Writes the line ascent in pixels to the `lascent` member of the `SCRIMG_WIN` struct: the line ascent is equal to the sum of `leading->top` and the font ascent.

Sets the modified window information by sending an `SI_SET` message to `edwin.scrimg`.

This method is expected to be followed by a call to the `ew_resize` method.

EW_LEAVE — Leave handling

```
VOID ew_leave(INT err);
```

Leave handling for the error code specified by `err`.

If `err` is `E_GEN_OVER`, calls `hBeep` and if `edwin.flags` contains `PR_EDWIN_NOTIFY_OVERFLOW`, calls `hInfoPrint` with an argument of `-SYS_EDIT_NCHARS`. Calls `f_leave` with an argument of `RUN_ACTIVE_CLEANUP_NONOTIFY`.

Otherwise calls `f_leave` with an argument of `err`.

EW_FIND — Search for text and show cursor

```
INT ew_find(TEXT *pstr, UINT flags);
```

Search the document for the zero terminated string specified by `pstr` ignoring strings that cross paragraph boundaries. The search starts at the selected region.

The search mode is specified by the ORed combination of flags in `flags`:

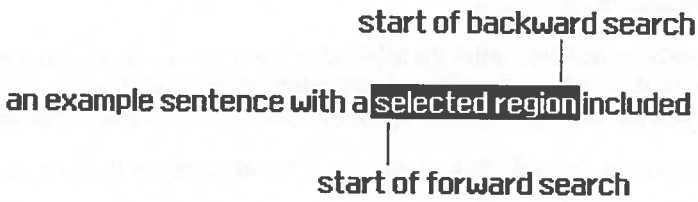
`EWF_CASESENS`	case sensitive matching: by default the matching is case insensitive.
`EWF_BACKWARDS`	search from the current cursor position moving towards the start of the document: by default the search moves towards the end of the document.

Senses the length and start position of the select region by sending an `SI_GET_SELECT` message to `edwin.scrimg`.

Senses the length of the document excluding the terminating zero by sending an `EP_SENSE_LEN` message to `edwin.doc`.

If no selected region exists, starts the search from the current cursor position.

If a selected region exists, starts the search from the position shown in the following picture:



Since the search selects the first matching string the above ensures that repeated searches do not locate the same matching string, i.e. the method effectively supports find next.

Starts the search for the first occurence of the required string.

On success selects the matching text moving the cursor to the beginning or end of the selected region depending on the search direction by sending `SI_MOVE_CURSOR` messages to `edwin.scrimg`. Returns `TRUE`.

On failure returns `FALSE`. The cursor position remains unchanged.

EW_REPLACE — Replace select

```
INT ew_replace(TEXT *replace, INT backwards);
```

Replace the text in the selected region with that specified by `replace`: the cursor is moved to the start (end) of the selected region if `backwards` is `TRUE` (`FALSE`).

If `edwin.flags` contains `PR_EDWIN_READONLY` *and* the return value from an `EW_READONLY` message is `TRUE`, calls `hBeep` then `p_leave` with `RUN_ACTIVE_USED` as argument.

Inserts the `replace` string before the selected region by sending an `EP_INSERT` message to `edwin.doc`: the message is sent using `p_entersend` and on error the method sends an `EW_LEAVE` message sent to `self` with the return value as argument.

Deletes the selected region by sending an `EP_DELETE` message to `edwin.doc`.

Cancels any selected region and moves the cursor to the end (start) of the inserted text if `backwards` is `FALSE` (`TRUE`) by sending an `SI_MOVE_CURSOR` message to `edwin.scrimg`.

Writes the cursor position to `edwin.cpos`. Writes `EW_CHANGE` to `edwin.change`.

Rebuilds the layout and redraws the view by sending an `SI_DOC_CHANGED` message to `edwin.scrimg`.

Confirms success by returning zero.

EW_EVALUATE — Evaluate an expression

```
VOID ew_evaluate(VOID);
```

Evaluate an expression: the expression is assumed to be selected or, if no region is selected, the cursor is assumed to indicate the position of an expression.

If `edwin.flags` contains `PR_EDWIN_READONLY` *and* the return value from an `EW_READONLY` message is `TRUE`, calls `hBeep` then call `p_leave` with `RUN_ACTIVE_USED` as argument.

Otherwise obtains the length and the position of the start of the selected region by sending an `SI_GET_SELECT` message to `edwin.scrimg`. If no region is selected attempts to select an expression as follows:

- moves the cursor towards the start of the document until it is positioned immediately before a word - i.e. a continuous sequence of characters containing no spaces or paragraph delimiters: the method sends an `EP_SCAN_WORD` message to `edwin.doc` and an `SI_MOVE_CURSOR` to `edwin.scrimg`.
- selects the word - i.e. the candidate expression - moves the cursor to the end of the word and scrolls the view until the cursor is visible: the method sends an `SI_MOVE_CURSOR` message to `edwin.scrimg` and an `EP_SCAN_WORD` message to `edwin.doc`.
- obtains the length and start of the word by sending an `SI_GET_SELECT` message to `edwin.scrimg`

If no region is selected displays the text message specified by the `SYS_NOTHING_TO_EVAL` system resource using the `hInfoPrint` utility routine. On English language machines this is "Nothing to evaluate". Returns.

If the selected region contains more than 254 characters, displays the text specified by the `SYS_TOOLONG_TO_EVAL` system resource using the `hInfoPrint` utility routine. On English language machines this is "Too long to evaluate". Return.

Copies the content of the selected region to a text buffer by sending an `EP_EXTRACT` message to `edwin.doc` and replaces all occurences of `0x00` with a space - i.e. ' '.

Evaluates the expression in the text buffer by sending a `WS_EVALUATE` message to `w_ws`.

If the return value is less than zero indicating successful evaluation:

- inserts at the current cursor position an equals sign followed by the result of the evaluation by sending an `EP_INSERT` message to edwin.doc. The message is sent using `p_entersend`: on error sends `self` an `EW_LEAVE` message with the return value as the argument.
- writes `EW_CHANGE` to `edwin.change` and rebuilds the layout and the view by sending an `SI_FWD_CHANGE` message to `edwin.scrimg`.
- selects the expression and the equals character and moves the cursor to the end of the selected region by sending `SI_MOVE_CURSOR` messages to `edwin.scrimg`. Writes the cursor position to `edwin.cpos`. The view is scrolled until the cursor is visible.

Otherwise:

- moves the cursor to the position in the expression where the error was detected by the `WS_EVALUATE` message and cancels any selected region by sending an `SI_MOVE_CURSOR` message to `edwin.scrimg`. The view is scrolled until the cursor is visible.

EW_REPLACE_CLIP — Copy select to clipboard

```
INT ew_replace_clip(VOID);
```

Replace the text in the clipboard with the text in the selected region.

Obtains the length and start of the selected region by sending an SI_GET_SELECT message to edwin.scrimg.

If no region is selected returns zero.

Deletes the content of the clipboard excluding the terminating zero by sending an EP_CLEAR message to edwin.clip.

Inserts the content of the selected region in the front of the clipboard by sending an EP_COPY_TO_FRONT message to edwin.doc.

Returns the number of characters copied.

EW_PASTE_CLIP — Paste in clipboard contents

```
INT ew_paste_clip(VOID);
```

Paste the content of the clipboard into the document at the current cursor position: the pasted text is selected and the cursor moved to the end of the selected region.

If edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep and then call p_leave with RUN_ACTIVE_USED as argument.

If the clipboard is empty returns FALSE - the length of the clipboard text excluding the terminating zero is sensed by sending an EP_SENSE_LEN message to edwin.clip.

Inserts the clipboard text into the document at the current cursor position and sets the cursor to the end of the inserted text by sending an EP_PASTE message to edwin.doc. Note that the message is sent using p_entersend: on error send self an EW_LEAVE message with the return value as the argument.

Writes the cursor position to edwin.cpos and writes EW_CHANGE to edwin.change. Rebuilds the layout and view as required by sending an SI_FWD_CHANGE message to edwin.scrimg.

Selects the inserted text maintaining the cursor at the end of the inserted text by sending SI_MOVE_CURSOR messages to edwin.scrimg.

Returns the number of characters inserted.

EW_BRING_IN — Perform bring

```
INT ew_bring_in(INT pid, INT format);
```

Insert into the document at the current cursor position one or more blocks of data of type format supplied by the process whose ID is pid.

The blocks of data may each contain up to and not exceeding 256 bytes of data.

The transaction can be terminated by sending a NULL block of data.

If edwin.flags contains PR_EDWIN_READONLY *and* the return value from an EW_READONLY message is TRUE, calls hBeep then p_leave with RUN_ACTIVE_USED as argument.

The following flag may be set in format:

EW_BRING_SINGLE_SHOT — specifies that the method should accept only one block of data containing up to 256 characters

The individual bits of format indicate the type of data: the bit index and the type of data are as follows:

DF_LINK_PARAS — insert each block of data without modification.

otherwise add a paragraph delimiter - 0x00 -to the end of each inserted block of data except the last.

DF_LINK_TABTEXT each block of data is ASCII text possibly containing tab characters.

DF_LINK_TEXT each block of data is plain ASCII text.

(The DF_LINK_TABTEXT and DF_LINK_TEXT data types are handled identically.)

Creates an instance of the LINKCL class and initialises by sending the LINKCL instance an LC_START message.

If format contains EW_BRING_SINGLE_SHOT, requests and receives one block of data and then sends an LC_STOP message to the link object.

Otherwise repeatedly requests and receives blocks of data until a NULL block of data is received.

The method handles each block of data as follows:

- gets the block of data by sending an LC_GET_DATA message to the link object.
- prepends an 0x00 paragraph delimiter as required - see above - and insert the data into the document by sending self an EW_EP_INSERT message.

N.B. If at any time the EW_EP_INSERT message returns a non-zero value - indicating an error - the method deletes all blocks of inserted data by sending an EP_DELETE message to edwin.doc and then sending self an EW_LEAVE message with as argument the non-zero return value.

Writes the length of the document to edwin.clen and writes EW_CHANGE to edwin.change.

Rebuilds the layout and view by sending an SI_FWD_CHANGE message to edwin.scrimg.

Selects the inserted text, moves the cursor to the end of the selected region and scrolls the view until the cursor is visible by sending SI_MOVE_CURSOR messages to edwin.scrimg. Writes the cursor position to edwin.cpos.

The PR_WSERV_RECEIVED_KEY flag is cleared in w_ws->wserv.flags.

Returns FALSE.

(Note that a more complete description of the link-paste mechanism can be found in the *Link Paste* chapter of the *Object Oriented Programming Guide*.)

EW_EP_INSERT — Insert text

```
INT ew_ep_insert(UINT pos, TEXT *buf, UINT blen);
```

Insert blen characters from the buffer specified by buf into the document at offset pos.

Inserts the text by sending an EP_INSERT message to edwin.doc.

If the application needs to take special note of the insertion of paragraph terminators, this method may be subclassed to deal with any zeros in the buffer (a simple strategy could be to send an EP_ADD_PARA message to edwin.doc for each zero-terminated segment of the buffer).

EW_RETURN_KEY — Return keypress handling

```
INT ew_return_key(INT shift);
```

Non-standard Return key handling.

The method is called from within the wn_key method - the default method returns FALSE indicating that Return keypresses are allowed.

Subclassers may replace this method to provide the desired functionality.

A subclass could for example constrain the document to contain a fixed number of lines and thus would return TRUE once the maximum number of lines had been reached.

EW_TAB_KEY — Tab keypress handling

```
INT ew_tab_key(INT shift);
```

Non-standard Tab key handling.

The method is called from within the `wn_key` method - the default method returns `FALSE` indicating that Tab keypresses are allowed.

Subclassers may replace this method to provide the desired functionality.

EW_INIT_STYLE — Customize style

```
VOID ew_init_style(SCRLAY_STYLE *pstyle);
```

The default method does nothing.

The method is called from within the `ew_init` method immediately before the style specified by `pstyle` is set into property: it can be replaced by subclassers in order to modify the default initialisation style.

EW_READONLY — Return read-only state

```
INT ew_readonly(VOID);
```

Return the read-only state of the current document.

The default method returns `TRUE`.

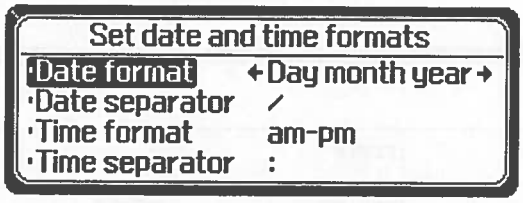
Subclassers may replace the method to provide the desired functionality.

A subclasser could for example allow read-only access while in outline mode and read-write access in normal mode.

PUNCTUED

WIN	LODGER	EDWIN		PUNCTUED
flags id	landlord offset width	scrimg scrlay doc cpos clen clip	select change flags margins font	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ wn_visible lg_draw lg_self_check ~~lg_set_id_pos~~ ~~wn_init~~ ~~lg_sense_width~~ *lg_update*	destroy wn_init ~~wn_key~~ wn_draw wn_sense_help wn_set wn_sense wn_emphasise lg_set_id_pos lg_sense_width ew_sense_size ew_set_size ew_set ew_sense ew_insert	ew_snuggle_insert ew_set_font ew_leave ew_find ew_replace ew_evaluate ew_replace_clip ew_paste_clip ew_bring_in ew_ep_insert ew_return_key ew_tab_key ew_init_style ew_readonly	wn_key

The `PUNCTUED` class provides an edit window displaying a single punctuation character. An example puctuation editor is shown in the following picture:



The punctuation editor is the fourth control in the above dialog and allows the user to alter the time separator character. The dialog is taken from the Time application.

Class definition

Defined in sub-category file *edwin.cl* (generated header file *edwin.g*).

```
CLASS punctued edwin
punctuation editor, one line editbox & it's a lodger
    {
    REPLACE wn_key
    }
```

Property

None.

PUNCTUED methods

WN_KEY — Handle key input

```
VOID wn_key(INT keycode,INT modifiers);
```

Handle a keypress.

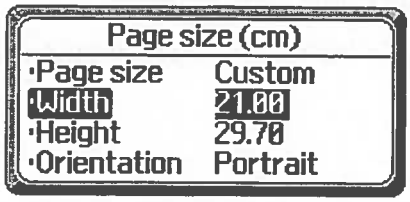
If `keycode` is not a punctuation character, calls `hBeep` and return.

Otherwise sets the punctuation character into the edit window by sending `self` a `WN_SET` message.

FLTEDIT

WIN	LODGER	EDWIN		FLTEDIT
flags	landlord	scrimg	select	current
id	offset	scrlay	change	low
	width	doc	flags	high
		cpos	margins	point
		clen	font	format
		clip		
~~destroy~~	~~destroy~~	destroy	ew_snuggle_insert	wn_init
wn_calc_position	wn_visible	~~wn_init~~	ew_set_font	wn_set
wn_connect	lg_draw	wn_key	ew_leave	wn_sense
wn_dodraw	~~lg_self_check~~	wn_draw	ew_find	lg_self_check
~~wn_emphasise~~	~~lg_set_id_pos~~	wn_sense_help	ew_replace	
~~wn_key~~	~~wn_init~~	~~wn_set~~	ew_evaluate	
wn_position		~~wn_sense~~	ew_replace_clip	
wn_redraw	~~lg_sense_width~~	wn_emphasise	ew_paste_clip	
~~wn_sense_help~~	lg_update	lg_set_id_pos	ew_bring_in	
~~wn_visible~~		lg_sense_width	ew_ep_insert	
		ew_sense_size	ew_return_key	
~~wn_set~~		ew_set_size	ew_tab_key	
~~wn_sense~~		ew_set	ew_init_style	
~~wn_draw~~		ew_sense	ew_readonly	
~~wn_init~~		ew_insert		

A floating point editor control presents an editable floating point number as shown in the following picture:



The second and third controls in the above dialog are both floating point editors with the display format set to fixed - i.e. a fixed number of decimal places.

Class definition

Defined in sub-category file *fltedit.cl* (generated header file *fltedit.g*).

```
CLASS fltedit edwin
floating point editor; works in either general or fixed format
    {
    REPLACE wn_init
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE lg_self_check

    CONSTANTS
        {
        FLTEDIT_MAX_CHAR        20 /* -x.xx...x (13 dec places) E-99 */
        SE_FLTEDIT_CURRENT      0x01
        SE_FLTEDIT_HIGH         0x02
        SE_FLTEDIT_LOW          0x04
        }
```

```
TYPES
    {
    typedef struct
        {
        DOUBLE current;  /* current value */
        DOUBLE low;      /* lower bound */
        DOUBLE high;     /* upper bound */
        UBYTE vulen;     /* view len, ie no. of char on display */
        UBYTE format;    /* fixed no of dec places, or 0 for general */
        } IN_FLTEDIT;
    typedef struct
        {
        DOUBLE current;   current value
        DOUBLE low;       lower bound
        DOUBLE high;      upper bound
        WORD set_flags;   which fields to set
        } SE_FLTEDIT;
    }

PROPERTY
    {
    DOUBLE current;
    DOUBLE low;
    DOUBLE high;
    UBYTE point;
    UBYTE format;   stores #decPlaces, which determine format (fixed or general)
    }
}
```

Property

`fltedit.current`	the editable value.
`fltedit.low`	the lower limit on the editable value.
`fltedit.high`	the upper limit on the editable value.
`fltedit.point`	the country specific decimal separator character: a full stop on English language machines.
`fltedit.format`	the number of digits after the decimal point or zero indicating general format.

FLTEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_FLTEDIT *init,PR_WIN *landlord);
```

Initialise the floating point editor according to the content of the `IN_FLTEDIT` struct pointed to by `init` and the ID of the landlord window specified by `landlord`.

Writes the country specific decimal separator character to `fltedit.point`.

The `IN_FLTEDIT` struct is defined as follows:

```
typedef struct
    {
    DOUBLE current;  /* current value */
    DOUBLE low;      /* lower bound */
    DOUBLE high;     /* upper bound */
    UBYTE vulen;     /* view len, ie no. of char on display */
    UBYTE format;    /* fixed no of dec places, or 0 for general */
    } IN_FLTEDIT;
```

The significance of the members of the IN_FLTEDIT struct is as follows:

current	the initial value for the editable value.
low	the lower limit on the editable value.
high	the upper limit on the editable value.
vulen	the maximum number of characters to represent the editable value in the edit window.
format	the number of digits after the decimal separator character or zero to indicate general format.

Sets the current value in the edit window by supersending a WN_INIT message.

WN_SET — Set values

```
VOID wn_set(SE_FLTEDIT *pset);
```

Set the content of the floating point editor according to the SE_FLTEDIT struct specified by pset.

The SE_FLTEDIT struct is defined as follows:

```
typedef struct
    {
    DOUBLE current;
    DOUBLE low;
    DOUBLE high;
    WORD set_flags;
    } SE_FLTEDIT;
```

The significance of the members of the SE_FLTEDIT struct is as follows:

current	the candidate editable value.
low	the candidate minimum allowed value.
high	the candidate maximum allowed value.
set_flags	see below.

The setting is controlled by ORing into the flags member of the SE_FLTEDIT struct a suitable combination of the following flags:

SE_FLTEDIT_CURRENT	indicates that the editable value should be set.
SE_FLTEDIT_HIGH	indicates that the upper limit should be set.
SE_FLTEDIT_LOW	indicates that the lower limit should be set.

Converts fltedit.current into a string representation and sets into the edit window by supersending a WN_SET message.

WN_SENSE — Sense current value

```
VOID wn_sense(DOUBLE *pdbl);
```

Sense the current value and write the result to the DOUBLE specified by pdbl.

Senses the content of the edit window by supersending a WN_SENSE message. Converts the string representation of the editable value into a double and writes the result to *pdbl.

LG_SELF_CHECK — Validate fields

```
INT lg_self_check(VOID);
```

Check the editable value.

On success return TRUE.

If the editable value is less than 1.0E-99 displays an appropriate error message using the hInfoPrintErr routine and returns E_GEN_UNDER.

If the editable value is greater than 1.0E99 displays an appropriate error message using the hInfoPrintErr routine and returns E_GEN_OVER.

If the editable value is not a number - 123abc for example - displays the SYS_INVALID_NUM system resource using the hInfoPrintErr utility routine and returns FALSE.

If the editable value is less than fltedit.low, writes fltedit.low to fltedit.current, displays an appropriate error message using the hInfoPrint utility routine - the error message is constructed from the SYS_OUT_OF_RANGE and SYS_MIN system resources and fltedit.low - and returns -1.

If the editable value is greater than fltedit.high, writes fltedit.low to fltedit.current, displays an appropriate error message using the hInfoPrint utility routine - the error message is constructed from the SYS_OUT_OF_RANGE and SYS_MAX system resources and fltedit.high - and returns 1.

Otherwise if no error detected returns TRUE.

XEDIT

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ wn_visible lg_draw lg_self_check ~~lg_set_id_pos~~ ~~wn_init~~ ~~lg_sense_width~~ *lg_update*

XEDIT
current data
destroy lg_sense_width wn_key wn_draw wn_set wn_sense wn_emphasise

The XEDIT class provides a single line text editer suitable for entering passwords of eight characters or less. An example password editor is shown in the following picture:



The characters typed in are indicated by the padlock symbols with the cursor position indicated by the solid black rectange.

Class definition

Defined in sub-category file *xedit.cl* (generated header file *xedit.g*).

```
CLASS   xedit     lodger
Secret editor with only 1 line of 8 chars to deal with keying in password.
    {
    REPLACE lg_sense_width
    REPLACE wn_key          process key press
    REPLACE wn_draw
    REPLACE wn_sense
    REPLACE wn_set          clears/blanks the data in property
    REPLACE wn_emphasise    always return false in this case

    CONSTANTS
        {
        XEDIT_MAX_LEN       8
        }

    TYPES
        {
        typedef struct
            {
            TEXT *pstr;
            } SE_XEDIT;
        }

    PROPERTY
        {
        UBYTE current;
        UBYTE width;
        TEXT data[XEDIT_MAX_LEN+2];
        }
    }
```

Property

`xedit.current`	the length of the password. *Warning: on the Series 3 this property is defined as a* `WORD`.
`xedit.width`	the width of a single character. *Warning: on the Series 3 this property is not defined.*
`xedit.data`	the password buffer containing the password - stored as a zero terminated string.

XEDIT methods

LG_SENSE_WIDTH — Get required width

```
INT lg_sense_width(VOID);
```

Sense the width of the password editor.

The method returns the width of the underscore character used in the password editor, multiplied by `(EDIT_MAX_LEN+1)`.

On the Series 3, on other machines running in compatibility mode and in a small font dialog on the Work*about*, the character is the normal underscore (ASCII `0x 5f`). In all other cases, the character is that specified by `WS_SYMBOL_PASS_UNDERLINE`, defined in *symbols.h*.

WN_KEY — Handle key input

```
INT wn_key(INT keycode,INT modifiers);
```

Handle a keypress.

If `keycode` is `W_KEY_DELETE_LEFT`, removes the last character from the password by writing zero to `xedit.data[xedit.current]` and decrementing `xedit.current`. Returns `WN_KEY_CHANGED`.

Otherwise if keycode is less than 0x00FF and the cursor position is less than XEDIT_MAX_LEN, adds the keypress to the password by writing keycode to xedit.data[xedit.current] and incrementing xedit.current. Returns WN_KEY_CHANGED.

Otherwise calls hBeep and returns WN_KEY_NO_CHANGE.

WN_DRAW — Draw

```
VOID wn_draw(VOID);
```

Draw the password editor.

Draws a WS_SYMBOL_PADLOCK character for each character of the password and an underscore character - as specified in the lg_sense_width method - for the remaining characters in the display.

If win.flags contains PR_WIN_EMPHASISED, draws a solid black rectangle to indicate the position of the cursor.

WN_SENSE — Sense data

```
VOID wn_sense(SE_XEDIT *psense);
```

Sense the content of the password editor and write the result to psense.

Writes the address of the first character in xedit.data to sense->pstr.

WN_SET — Clear password data

```
VOID wn_set(VOID);
```

Reset the password.

Clears the password by writing zero to each element of xedit.data and writing zero to xedit.current.

Draws the control by sending self an LG_DRAW message.

WN_EMPHASISE — Set emphasis

```
VOID wn_emphasise(INT flags);
```

Emphasise the control if flags is non-zero, otherwise de-emphasise the control.

If flags is non-zero, sets PR_WIN_EMPHASISED in win.flags.

Otherwise clears PR_WIN_EMPHASISED in win.flags.

Sends self an LG_DRAW message.

CHAPTER 11

GAUGE CLASSES

This chapter describes the GAUGE and DONEWN classes which support graphical gauge controls for use as components in a dialog.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the WIN and LODGER classes described in the *Windows* chapter of the *HWIM Reference* manual.
- the DLGBOX class described in the *Dialog Boxes* chapter of the *HWIM Reference* manual.

Class diagram

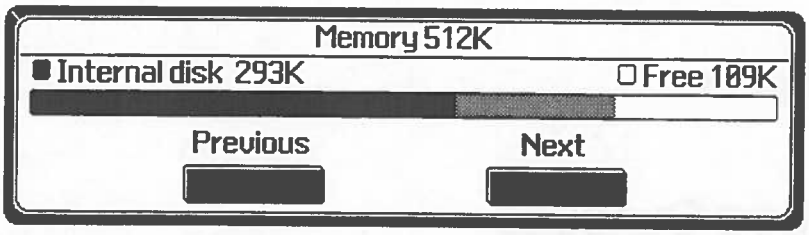


GAUGE

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw wn_emphasise wn_key wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ *wn_sense* ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw lg_self_check lg_set_id_pos ~~*lg_sense_width*~~ *lg_update*

GAUGE
size colour
wn_draw wn_init wn_set lg_sense_width

The GAUGE class provides a graphical representation of a gauge with four segments each of which may be either, black, grey or white. An example of an instance of the GAUGE class used as a component in a dialog is shown in the following picture:



The above example uses three segments to indicate the usage of memory: the width of the fourth segment is set to zero. This dialog may be obtained by selecting the *Memory info* menu item in the system screen.

See also the DONEWN class described in a later section of this chapter.

Class definition

Defined in sub-category file *gauge.cl* (generated header file *gauge.g*).

```
CLASS   gauge    lodger
Gauge Display with 3 colour graphical representation
    {
    REPLACE wn_init          Set item Dead and centre & setup landlord
    REPLACE wn_set           Sets up the colour and size of each section of  gauge
    REPLACE lg_sense_width   Return min possible width for gauge
    REPLACE wn_draw          Draws the gauge

    CONSTANTS
        {
        GAUGE_WHITE              1
        GAUGE_GREY               2
        GAUGE_BLACK              3
        }

    TYPES
        {
        typedef struct
            {
            UWORD size[4];
            UBYTE colour[4];
            } SE_GAUGE;
        }

    PROPERTY
        {
        UWORD size[4];
        UBYTE colour[4];
        }
    }
```

Property

`gauge.size` an array of four elements each of which defines the width of a gauge segment.

`gauge.colour` an array of four elements each of which defines the colour of a gauge segment. Available colours are GAUGE_WHITE, GAUGE_GREY and GAUGE_BLACK.

GAUGE methods

WN_INIT — Initialise

```
VOID wn_init(SE_GAUGE *par,PR_WIN *landlord);
```

Initialise the gauge to appear as a dead and centred component in the window with id `landlord`.

Writes `landlord` to `lodger.landlord` and sets DLGBOX_ITEM_CENTRE and DLGBOX_ITEM_DEAD in the landlord flags property by sending a DL_SET_ITEM_FLAGS message to `lodger.landlord`.

The `par` argument is included for use by subclassers: a replacement method might, for example, supersend a WN_INIT message and then send a WN_SET message with initial values for the segment widths and colours as specified in the SE_GAUGE struct.

WN_SET — Set characteristics of gauge sections

```
VOID wn_set(SE_GAUGE *pset);
```

Set the appearance of the gauge according to the content of the SE_GAUGE struct with address pset.

Scales the segment sizes specified in the SE_GAUGE struct such that the gauge has width lodger.width. Writes the scaled sizes and the colours from the SE_GAUGE struct to the gauge.size and gauge.colour property respectively. Draws the lodger window by sending an LG_DRAW message to self.

LG_SENSE_WIDTH — Return minimum width for gauge

```
INT lg_sense_width(VOID);
```

Return the minimum width for the gauge.

WN_DRAW — Draw the gauge

```
VOID wn_draw(VOID);
```

Draw the gauge component in the lodger window.

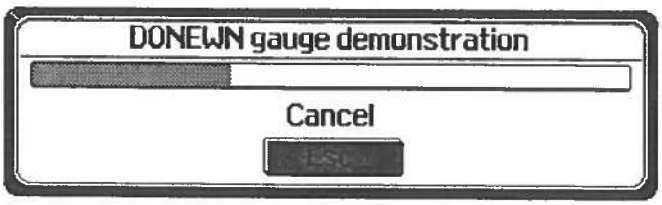
The gauge has height eight pixels on the Series 3 and thirteen pixels on the Series 3a and width lodger.width on both machines. The relative widths and the colours of the gauge segments are as specified by the last WN_SET message.

DONEWN

WIN	LODGER	GAUGE
flags id	landlord offset width	size colour
~~destroy~~ wn_calc_position wn_connect wn_dodraw wn_emphasise wn_key wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ *wn_sense* ~~wn_draw~~ ~~wn_init~~	destroy ~~wn_init~~ wn_visible lg_draw lg_self_check lg_set_id_pos ~~*lg_sense_width*~~ *lg_update*	wn_draw wn_init ~~wn_set~~ lg_sense_width

DONEWN
val range
wn_set

The DONEWN class supports a graphical gauge that can be used as a component in a dialog or in a compatible subclass of the LODGER class. An example of a DONEWN gauge is shown in the following picture:



Further examples can be obtained by selecting the *Format disk* and *Copy disk* commands from the Series 3 system menu.

The dialog shown in the picture above can be created using the following resources:

```
RESOURCE ACLIST_ARRAY replace_ac

    {
    button=
        {
        PUSH_BUT
            {
            keycode=W_KEY_ESCAPE;
            str="Cancel";
            }
        };
    }

RESOURCE DIALOG replace_dl_res
    {
    title="DONEWN gauge demonstration";
    flags=DLGBOX_NOTIFY_ESCAPE|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_DONEWN;
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=replace_ac;
                };
            }
        };
    }
```

Class definition

Defined in sub-category file *donewn.cl* (generated header file *donewn.g*).

```
CLASS   donewn    gauge
    {
    REPLACE wn_set
    CONSTANTS
        {
        SE_DONEWN_RANGE       0x01
        SE_DONEWN_VALUE       0x02
        SE_DONEWN_INCREMENT   0x04  /* val will be increased by 1 */
        SE_DONEWN_INC_VAL     0x08  /* val will be increased by whatever val is set */
        }

    TYPES
        {
        typedef struct
            {
            UWORD flags;
            ULONG val;
            ULONG range;
            } SE_DONEWN;
        }

    PROPERTY
        {
        ULONG val;
        ULONG range;
        }
    }
```

Property

`donewn.val`	specifies the relative width of the first segment. This segment has colour grey.
`donewn.range`	specifies the relative width of the gauge display. The second segment has colour white and relative width `range-val`.

DONEWN methods

WN_SET — Set gauge

```
VOID wn_set(SE_DONEWN *pset);
```

Set the gauge according to the content of the `SE_DONEWN` struct with address `pset`.

The `SE_DONEWN` struct is defined as follows:

```
typedef struct
    {
    ULONG flags
    ULONG val
    ULONG range
    } SE_DONEWN;
```

The property to be set is specified by ORing one or more of the following flags into the `flags` member of the above struct.

`SE_DONEWN_RANGE`	indicates that `donewn.range` should be set from member `range`
`SE_DONEWN_VALUE`	indicates that `donewn.val` should be set from member `val`
`SE_DONEWN_INCREMENT`	indicates that `donewn.val` should be incremented by one unit
`SE_DONEWN_INC_VALUE`	indicates that `donewn.val` should be incremented by member `val`

Note that `donewn.val` is assumed to be less than or equal to `donewn.range`. Whenever this is not the case, it will be set equal to `donewn.range`.

For the third and fourth segments, writes zero to the size and `GAUGE_BLACK` to the colour property. These segments will thus not be visible.

Sets the values into the superclass property by supersending a `WN_SET` message.

Using Gauges

The following section includes examples of dialogs containing `GAUGE` and `DONEWN` controls. See also the gauge example code that can be installed from the SDK Optional disk into a *\sibosdk\hwimdemo* directory.

A GAUGE class example

The first example illustrates the use of the `GAUGE` class as a dialog component. The example dialog simply contains a title and a `GAUGE` control.

The initial appearance of the `GAUGE` control is set by passing an appropriately initialised `SE_GAUGE` struct to the dialog on creation.

The structure and content of the dialog is specified using the following dialog resource:

```
RESOURCE DIALOG gauge_dl_res
    {
    title="GAUGE demonstration";
    flags=DLGBOX_NOTIFY_ESCAPE|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_GAUGE;
            }
        };
    }
```

The GAUGEDLG dialog class subclasses DLGBOX and replaces the `dl_dyn_init` method to allow dynamic initialisation of the dialog. The GAUGEDLG class is defined as follows:

```
CLASS gaugedlg dlgbox
    {
    REPLACE dl_dyn_init;
    }
```

The code for the `dl_dyn_init` method is as follows:

```
METHOD VOID gaugedlg_dl_dyn_init(PR_GAUGEDLG *self)
    {
    p_send4(self,WN_SET,1,self->dlgbox.rbuf);
    }
```

The `dl_dyn_init` method does no more than set the GAUGE control using the SE_GAUGE struct pointed to by `self->dlgbox.rbuf`.

The following code may be used to launch the dialog:

```
LOCAL_C INT LaunchDial(VOID *buf,INT id,INT class)
    {
    DL_DATA dl_data;

    dl_data.id=id;
    dl_data.rbuf=buf;
    dl_data.pdlg=NULL;
    return(p_send5(w_ws,O_WS_DO_DIAL,p_getlibh(CAT_DEMO_DEMO),class,&dl_data));
    }

SE_GAUGE se_gauge;

/* set the members of se_gauge as appropriate */
...
LaunchDial(&se_gauge,GAUGE_DL_RES,C_GAUGEDLG);
...
```

A DONEWN class example

The second example illustrates using the DONEWN class as a dialog control to inform the user of memory usage.

The initial appearance of the DONEWN control is specified by means of an SE_DONEWN struct passed to the dialog on creation.

The structure and content of the dialog is specified by the following dialog resource:

```
RESOURCE DIALOG gauge_dl_res
    {
    title="DONEWN demonstration";
    flags=DLGBOX_NOTIFY_ESCAPE|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_DONEWN;
            }
        };
    }
```

The DONEDLG dialog class subclasses DLGBOX and replaces the dl_dyn_init method to provide application specific initialiation. The GAUGEDLG class and the dl_dyn_init method are defined as follows:

```
CLASS donedlg dlgbox
    {
    REPLACE dl_dyn_init;
    }

RESOURCE STRING
    {
    str="Memory used is %d Kbytes";
    }

METHOD VOID donedlg_dl_dyn_init(PR_DONEDLG *self)
    {
    TEXT buf[DONEDLG_MAX_TITLE];

    hAtob(&buf[0],DONEDLG_TITLE_RES,self->dlgbox.rbuf->val);
    hDlgSetText(0,&buf[0]);
    p_send4(self,WN_SET,1,self->dlgbox.rbuf);
    }
```

The dl_dyn_init method initialises the text in the title and the segment in the gauge according to the content of the SE_DONEWN struct pointed to by dlgbox.rbuf.

The dialog may be launched as follows:

```
SE_DONEWN se_donewn;

/* set the members of se_donewn as appropriate */
...
LaunchDial(&se_donewn,GAUGE_DL_RES,C_GAUGEDLG)
...
```

where the LaunchDial utility routine is defined in the previous section.

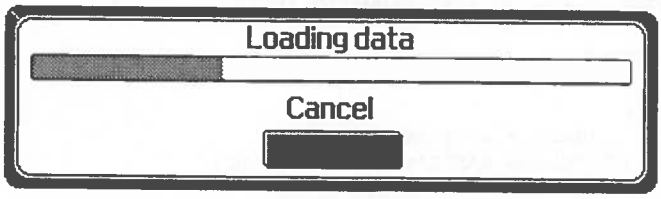
A dynamic gauge

A common use of the DONEWN class is to inform the user of the progress of a lengthy operation: the length of the shaded region of the gauge - which is initially set to zero - is simply incremented after the completion of each step of the operation.

This may be done by simply queueing a suitable active object before launching the dialog by sending a WS_DO_DIAL message to the WSERV object. The active object will run as soon as the dialog is visible: the WS_DO_DIAL message causes an AM_START message to be sent to the application manager thus giving an opportunity to all queued active objects to run. This opportunity to run is cancelled by the dialog when on terminating it sends an AM_STOP message to the application manager.

In the example, a lengthy operation is simulated by an active object that does no more than queue a timer and then update the gauge control in the dialog.

An example dynamic gauge is shown in the following picture:



The active object and the dialog are launched as follows:

```
ULONG duration;

duration=100; /* tenths of a second */
f_newsend(CAT_DUMP_DUMP,C_ATIMER,O_AO_INIT,&duration);
LaunchDial(&duration,GAUGE_DL_RES,C_GAUGEDLG);
```

where duration specifies the length of the operation in tenths of a second.

The initial content and appearance of the dialog are specified by means of the following resources:

```
RESOURCE ACLIST_ARRAY escape_ac
    {
    button=
        {
        PUSH_BUT
            {
            keycode=W_KEY_ESCAPE;
            str="Cancel";
            }
        };
    }

RESOURCE DIALOG gauge_dl_res
    {
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                str="Loading data";
                };
            },
        CONTROL
            {
            class=C_DONEWN;
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=escape_ac;
                };
            }
        };
    }
```

The GAUGEDLG class subclasses DLGBOX and replace the `dl_dyn_init` method to allow dynamic initialisation of the dialog. The GAUGEDLG class is defined as follows:

```
CLASS gaugedlg dlgbox
    {
    REPLACE dl_dyn_init
    }
```

The code for the `dl_dyn_init` method is as follows:

```
METHOD VOID gaugedlg_dl_dyn_init(PR_GAUGEDLG *self)
    {
    SE_DONEWN se_donewn;

    se_donewn.val=0L;
    se_donewn.range=*(ULONG *)self->dlgbox.rbuf;
    se_donewn.flags=SE_DONEWN_RANGE|SE_DONEWN_VALUE;
    p_send4(self,O_WN_SET,1,&se_donewn);
    }
```

The `dl_dyn_init` method does no more than set the initial value and the range of the DONEWN control.

The ATIMER class provides the active object which updates the gauge every one tenth of a second. After the elapse of a predetermined interval the active object destroys both itself and the dialog. The ATIMER class is defined as follows:

```
CLASS atimer timer
    {
    REPLACE ao_init
    REPLACE ao_run
    REPLACE ao_queue
    PROPERTY
        {
        ULONG count;
        ULONG total;
        }
    }
```

The code for the ao_queue method is as follows:

```
METHOD VOID atimer_ao_queue(PR_ATIMER *self,UINT lsw,UINT msw)
    {
    self->active.stat=E_FILE_PENDING;
    self->active.isactive=TRUE;
    p_supersend4(self,O_AO_QUEUE,lsw,msw);
    }
```

The ao_queue method queues a timer the duration of which is specified as a long integer broken down into the least significant word - lsw - and the most significant word - msw.

The code for the ao_init method is as follows:

```
METHOD VOID atimer_ao_init(PR_ATIMER *self,ULONG *ptotal)
    {
    p_send3(w_am,O_AM_ADD_TASK,self);
    self->active.priority=PRIORITY_ACTIVE_WSERV-1;
    self->atimer.count=0L;
    self->atimer.total=*ptotal;
    p_supersend2(self,O_AO_INIT);
    p_send4(self,O_AO_QUEUE,1,0);
    }
```

The ao_init method carries out the following actions:

- adds self to the task queue, and sets the task priority to PRIORITY_ACTIVE_WSERV less one, to ensure that the timer does not interfere with keypresses.
- sets the initial count and stores the total duration.
- opens a timer channel by supersending an AO_INIT message and then queues a timer.

The code for the ao_run method is as follows:

```
METHOD INT atimer_ao_run(PR_ATIMER *self)
    {
    SE_DONEWN se_donewn;

    if (w_ws->wserv.dial)
        {
        self->atimer.count++;
        se_donewn.flags=SE_DONEWN_INCREMENT;
        p_send4(w_ws->wserv.dial,O_WN_SET,1,&se_donewn);
        if (self->atimer.count<self->atimer.total)
            {
            p_send4(self,O_AO_QUEUE,1,0);
            return(RUN_ACTIVE_USED);
            }
        else
            p_send2(w_ws->wserv.dial,O_DESTROY);
        }
    p_send2(self,O_DESTROY);
    return(RUN_ACTIVE_USED);
    }
```

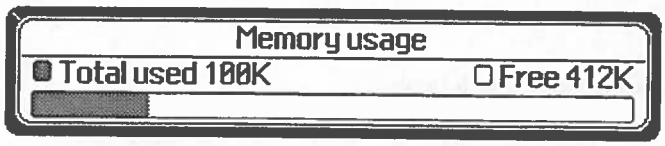
The ao_run method carries out the following actions:

- if the dialog does not exist - i.e the user has pressed the Escape key - it sends self a DESTROY message and returns RUN_ACTIVE_USED.

- if the maximum allowed number of timers has been reached, it sends DESTROY messages to `self` and to the dialog and then returns `RUN_ACTIVE_USED`.
- otherwise it increments the GAUGE control display and queues the next timer.

Annotating a gauge

The following section illustrates a convenient means of labelling a gauge control The following example dialog indicates the current memory usage:



The second item is a TEXTWIN control with appropriate text set for the prompt and the body text.

On the S3a the following resource may be used to create the above dialog:

```
RESOURCE STRING used_kbytes_res
    {
    str="%c Total used %luK";
    }

RESOURCE STRING free_kbytes_res
    {
    str="%c Free %luK";
    }

RESOURCE DIALOG demo_dl_res
    {
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    title="Memory usage";
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_DEAD;
            prompt=" ";
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_RIGHT;
                };
            },
        CONTROL
            {
            class=C_DONEWN;
            }
        };
    }
```

Note that the prompt for the TEXTWIN control is defined as a space in order that the prompt may be redefined dynamically.

The DEMODLG class subclasses the DLGBOX class and replaces the `dl_dyn_init` method to allow dynamic initialisation of the dialog. The DEMODLG class is defined as follows:

```
CLASS demodlg dlgbox
    {
    REPLACE dl_dyn_init
    }
```

The code for the `dl_dyn_init` method is as follows:

```
METHOD VOID demodlg_dl_dyn_init(PR_DEMODLG *self)
    {
    TEXT buf[40],resbuf[40];
    ULONG free;
    ULONG used;

    p_send4(self,O_WN_SET,2,(SE_DONEWN *)self->dlgbox.rbuf);
    used=((SE_DONEWN *)self->dlgbox.rbuf)->val;
    p_send4(w_am,O_AM_LOAD_RES_BUF,USED_KBYTES_RES,&resbuf[0]);
    p_atos(&buf[0],&resbuf[0],WS_SYMBOL_GREY_BOX,used);
    hDlgSetPrompt(1,&buf[0]);
    free=((SE_DONEWN *)self->dlgbox.rbuf)->range-used;
    p_send4(w_am,O_AM_LOAD_RES_BUF,FREE_KBYTES_RES,&resbuf[0]);
    p_atos(&buf[0],&resbuf[0],WS_SYMBOL_WHITE_BOX,free);
    hDlgSetText(1,&buf[0]);
    }
```

The method carries out the following actions:

- sets the DONEWN control using the SE_DONEWN struct pointed to by dlgbox.rbuf.
- sets an appropriate prompt for the TEXTWIN control indicating the memory usage.
- sets an appropriate body text for the TEXTWIN control indicating the free memory.

The following code may be used to run the dialog:

```
METHOD VOID democom_com_run(PR_DEMOCOM *self)
    {
    SE_DONEWN se_donewn;

    se_donewn.val=100L;
    se_donewn.range=512L;
    se_donewn.flags=SE_DONEWN_VALUE|SE_DONEWN_RANGE;
    LaunchDial(&se_donewn,DEMO_DL_RES,C_DEMODLG);
    }
```

where the hLaunchDial utility routine is defined in an earlier section.

On the Series 3 boxes may be constructed as follows:

- a grey box may be constructed by concatenating the WS_SYMBOL_GREY_BOX1 and WS_SYMBOL_GREY_BOX2 characters.
- a white box may be constructed by concatenating the WS_SYMBOL_WHITE_BOX1 and WS_SYMBOL_WHITE_BOX2 characters.
- a black box may be constructed by concatenating the WS_SYMBOL_BLACK_BOX1 and WS_SYMBOL_BLACK_BOX2 characters.

CHAPTER 12

FILE SELECTORS

This chapter documents some of the classes associated with the filing system. These classes are:

- the VADEV class which creates and stores a list of the available devices.
- the PACKSEL class which provides a dialog control that allows the user to select a pack.
- the FNEDIT class which provides a dialog control that allows the user to edit a path and a filename.
- and the FNSELWN class which provides a dialog control that allows the user to select an existing file.

For a description of the FILELIST class - which supports the graphical file list - see the *FILELIST* chapter of the *HWIM Reference* manual.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the VAROOT and related classes described in the *Variable Array Classes* chapter of the *OLIB Reference* manual.

VADEV

VAROOT		VASTR
nrec	key	size gran base len recno precno
destroy va_count va_delete va_sort va_key va_findisq va_insertisq va_append va_insert ~~va_search~~ va_compare va_reset va_test	va_replace ~~va_copy~~ ~~va_reclen~~ va_swap va_init ~~va_deletem~~ ~~va_insertm~~ ~~va_prec~~ ~~va_pbuf~~ ~~va_capacity~~ ~~va_compress~~	va_copy va_reclen ~~va_init~~ va_compress va_capacity va_deletem va_insertm va_prec ~~va_pbuf~~

VADEV
buf
va_init va_search va_pbuf

The VADEV class may be used to create a list of available devices stored as variable length text records. The class provides methods to search for a device and to retrieve the public device name.

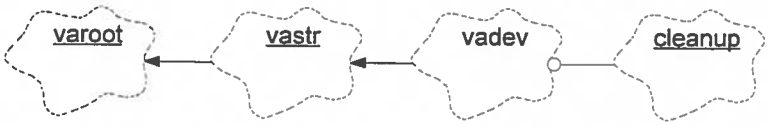
The list may be updated by simply creating a new instance of the class.

An application that wishes to use the VADEV class should ensure that:

- the application manager creates an instance of the CLEANUP class during initialisation.
- the application manager property is stored in the w_am magic static.

Both of the above conditions are satisfied by the HWIMMAN application manager class.

Class diagram



Class definition

Defined in sub-category file *files.cl* (generated header file *files.g*).

```
CLASS vadev     vastr
    {
    REPLACE va_init
    REPLACE va_search
    REPLACE va_pbuf
    PROPERTY
        {
        TEXT buf[20];
        }
    }
```

Property

`vadev.buf` contains the public device specification written by the last VA_PBUF message. If a VA_PBUF message has not been sent since instantiation the buffer contains the NULL string.

VADEV methods

VA_INIT — Initialise

```
VOID va_init(VOID);
```

Scan the available devices and for each create a record containing a device specification with format *node*::*device*:.

The first record specifies the local internal device: `LOC::M:`.

VA_SEARCH — Search for a device

```
INT va_search(TEXT *target);
```

Return the index of the record that matches the device specification pointed to by `target`.

The device specification pointed to by target may omit the trailing colon thus obeying the format *node*::*device*. This is not recommended as it increases the time required to locate the record.

On failure return 1.

VA_PBUF — Return public device specification

```
TEXT *va_pbuf(UWORD num);
```

Return a pointer to the public name of the device specified in record `num`.

For the local internal device the public name is read from the SYS_DEVICE_MEMORY system resource. On English language machines it is "Internal".

For all other local devices the public name for a device with specification LOC::*device*: is *device*. (Thus the public form of LOC::A: is A.)

For all other devices the public name is identical to the device specification.

PACKSEL

WIN	LODGER	CHLIST
flags id	landlord offset width	data flags matcher matchlen nsel pop
~~destroy~~ wn_calc_positio n wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ *lg_update*	destroy lg_sense_width wn_draw wn_emphasise ~~wn_init~~ ~~wn_key~~ wn_sense ~~wn_set~~

PACKSEL
filsel err
wn_init wn_set wn_key lg_self_check

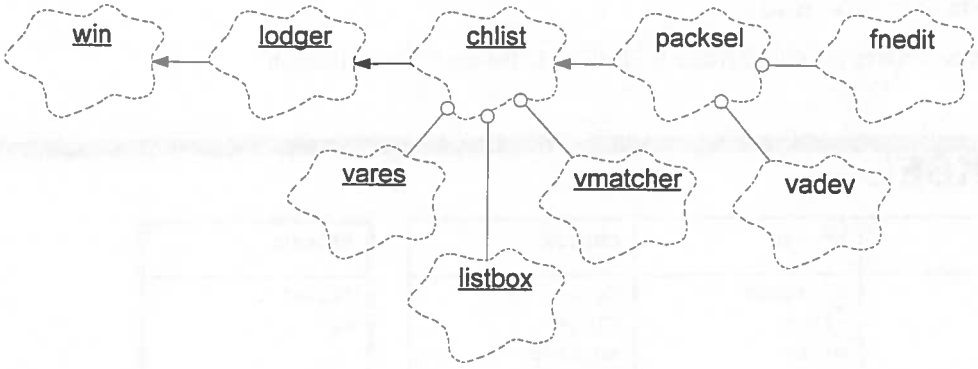
The PACKSEL class implements the pack selector control which allows the user to select the required device from the available list. The selection may be changed using the left and right arrow keys, first letter matching, or by pressing the Tab key to obtain a pop-out menu. An example pack selector control is shown in the following picture:



Both the file name editor and file name selector controls may include a pack selector simply by ORing the DLGBOX_ITEM_NEEDS_PACK flag into the flags member of the CONTROL resource as follows:

```
RESOURCE DIALOG demonstration
    {
    title="Open .pic file";
    flags=DLGBOX_NOTIFY_ENTERDLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_FNSELWN;
            flags=DLGBOX_ITEM_NEEDS_PACK;
            prompt="";
            info=FNSELWN
                {
                fname="Dtedit.pic";
                };
            }
        };
    }
```

Class diagram



Class definition

Defined in sub-category file *files.cl* (generated header file *files.g*).

```
CLASS packsel   chlist
    {
    REPLACE wn_init             build data to pass on to chlist
    REPLACE wn_set              convert text to nsel item
    REPLACE wn_key              notify filsel of any changes
    REPLACE lg_self_check       checks whether drive is empty

    CONSTANTS
        {
        PR_CHLIST_PACKSEL_DODINFO       0x8000
        }

    PROPERTY
        {
        PR_LODGER *filsel;      fnedit or fnselwn
        WORD err;               result of last dinfo on self
        }
    }
```

Property

`packsel.filsel` this *must* contain the handle of an instance of either the FNEDIT class or the FNSELWN class.

`packsel.err` this is a code giving the error status of the current device. It is obtained by calling the `p_dinfo` PLIB routine.

PACKSEL methods

WN_INIT — Initialise

```
VOID wn_init(TEXT *init,PR_WIN *landlord,PR_LODGER *filsel);
```

Initialise the pack selector according to the landlord window ID specified by `landlord` and the handle of an instance of either the FNEDIT class, or the FNSELWN class, specified by `filsel`.

Writes `landlord` to `lodger.landlord`.

Writes `filsel` to `packsel.filsel`.

The class uses a VADEV component to store the names of the available devices: creates an instance of the VADEV class, writes the handle to `chlist.data` and then initialises the VADEV component by sending a VA_INIT message to `chlist.data`.

Note that the `init` argument is not used.

WN_SET — Set pack from file name

```
INT wn_set(TEXT *pset,P_FPARSE *pcrk);
```

Set the current pack.

The `pset` and `pcrk` arguments are assumed to point to a full file specification and an associated `P_FPARSE` struct respectively: sets the current device according to the device specified by the arguments.

If `chlist.flags` contains `PR_CHLIST_PACKSEL_DODINFO`:

- obtains an error code for the current device by calling the `p_dinfo` `PLIB` routine and writing the return value to `packsel.err`.

Returns `packsel.err`.

WN_KEY — Handle key input

```
INT wn_key(INT keycode,INT modifierss);
```

Handle a keypress.

Allows the superclass to handle the keypress by supersending a `WN_KEY` message.

If the return value from the `WN_KEY` message is `WN_KEY_CHANGED` - in which case the current device has changed - displays the information message in the `SYS_SCANNING` system resource: on English language machines this is "Scanning".

If `chlist.flags` contains the `PR_CHLIST_PACKSEL_DODINFO` flag:

- obtains an error code for the current device by calling the `p_dinfo` `PLIB` routine and writing the return value to `packsel.err`.

Ensures that keypresses are not absorbed by the control by writing `FALSE` to `dlgbox.absorb`.

Sets the current device for the associated `FNEDIT` or `FNSELWN` control by sending an `LG_UPDATE` message to `packsel.filsel` passing as arguments a pointer to the current device specification, and an error status: if `chlist.flags` contains `PR_CHLIST_PACKSEL_DODINFO`, the error status is `packsel.err`, and zero otherwise.

Ensures that any information message is cancelled.

Returns the return value from the `WN_KEY` message.

LG_SELF_CHECK — Validate drive

```
INT lg_self_check(VOID);
```

Validate the current device.

If `packsel.err` is non-zero, or `PR_CHLIST_PACKSEL_DODINFO` is set in `chlist.flags`, obtains an error code for the current device by calling the `p_dinfo` `PLIB` library routine and then writes the error code to `packsel.err`.

If `packsel.err` is still non-zero, displays an appropriate information message, and then returns `FALSE`.

If `packsel.err` was previously non-zero:

- displays the information message in the `SYS_SCANNING` system resource: on English language machines this is "Scanning".
- sets the current device for the associated `FNEDIT` or `FNSELWN` control by sending an `LG_UPDATE` message to `packsel.filsel`.
- ensures that the information message is cancelled and then returns `TRUE`.

FNEDIT

WIN	LODGER	EDWIN		FNEDIT
flags id	landlord offset width	scrimg scrlay doc cpos clen clip	select change flags margins font	pop pack flags defext extbuf buf
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ ~~wn_init~~ ~~wn_visible~~ lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ lg_update	destroy ~~wn_init~~ ~~wn_key~~ wn_draw wn_sense_help ~~wn_set~~ ~~wn_sense~~ ~~wn_emphasise~~ lg_set_id_pos lg_sense_width ew_sense_size ew_set_size ew_set ew_sense ew_insert	ew_snuggle_insert ew_set_font ew_leave ew_find ew_replace ew_evaluate ew_replace_clip ew_paste_clip ew_bring_in ew_ep_insert ew_return_key ew_tab_key ew_init_style ew_readonly	wn_init wn_set wn_sense wn_key wn_emphasise lg_self_check lg_update

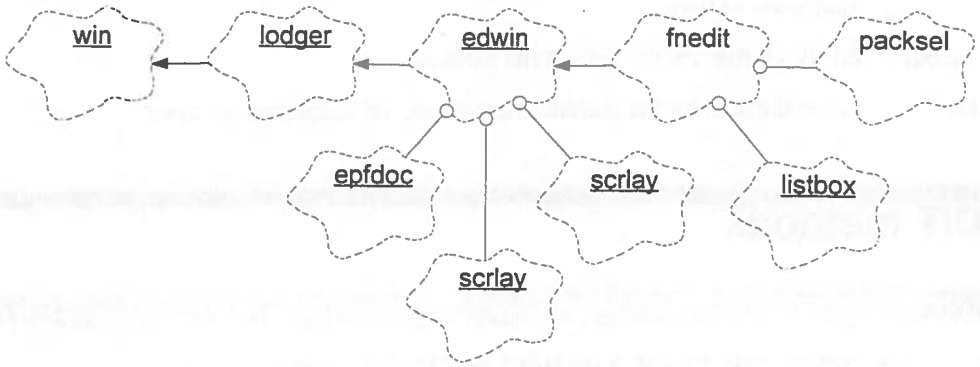
The FNEDIT class implements the file name editor control which allows the user to edit a file name. An example file name editor is shown in the following picture:



A file name editor may have an associated pack selector control by ORing the DLGBOX_ITEM_NEEDS_PACK flag into the flags member of the FNEDIT resource as follows:

```
RESOURCE DIALOG demonstration
    {
    title="Open .pic file";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_FNSELWN;
            flags=DLGBOX_ITEM_NEEDS_PACK;
            prompt="";
            info=FNSELWN
                {
                fname="Dtedit.pic";
                };
            }
        };
    }
```

Class diagram



Note that an instance of the packsel class is optional.

Class definition

Defined in sub-category file *files.cl* (generated header file *files.g*).

```
CLASS fnedit    edwin
    {
    REPLACE wn_init
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_key
    REPLACE wn_emphasise
    REPLACE lg_self_check
    REPLACE lg_update

    CONSTANTS
        {
        IN_FNEDIT_STANDARD      H_FILE_STANDARD_INIT
        IN_FNEDIT_ALLOW_DIRS    H_FILE_ALLOW_DIRS
        IN_FNEDIT_JUST_DIRS     H_FILE_JUST_DIRS
        IN_FNEDIT_FORCE_NXIST   H_FILE_FORCE_NXIST
        IN_FNEDIT_NO_AUTOQUERY  H_FILE_NO_AUTOQUERY
        IN_FNEDIT_ACCEPT_NULL   H_FILE_ACCEPT_NULL
        IN_FNEDIT_SET_DEFEXT    H_FILE_SET_DEFEXT
        IN_FNEDIT_CAN_WILDCARD  H_FILE_CAN_WILDCARD
        }

    TYPES
        {
        typedef struct
            {
            UBYTE flags;
            TEXT fname[1];
            } IN_FNEDIT;
        }

    PROPERTY 1
        {
        PR_LISTBOX *pop; handle of pop-out menu (NULL if un-squirted mode)
        PR_PACKSEL *pack;
        UWORD flags;
        TEXT *defext;
        TEXT extbuf[6];
        TEXT buf[P_FNAMESIZE];
        }
    }
```

Property

`fnedit.pop` this is either the handle of an instance of the FILELIST class - in which case the user has pressed the Tab key - or NULL otherwise.

`fnedit.pack` this is either the handle of an instance of the PACKSEL class or NULL if there is no associated pack selector control.

`fnedit.flags` an ORed combination of flags that specify the behaviour (see below)

fnedit.defext	this is a pointer to the default file extension and points to either fnedit.buf or w_am->hwimman.defext.
fnedit.extbuf	this is a buffer for the default file extension.
fnedit.buf	this is the path for the current file, e.g. REM::E:\SIBOSDK\OOPDEMO\.

FNEDIT methods

WN_INIT — Initialise

```
VOID wn_init(IN_FNEDIT *par,PR_WIN *landlord,PR_PACKSEL *pack);
```

Initialise the file name editor according to the content of par->flags.

Writes pack to fnedit.pack.

If par->flags contains IN_FNEDIT_JUST_DIRS, ensures that the remaining flags are valid: sets IN_FNEDIT_ALLOW_DIRS, and clears IN_FNEDIT_SET_DEFEXT, IN_FNEDIT_CAN_WILDCARD and H_FILE_CAN_TAG.

Writes par.flags to fnedit.flags.

Writes w_am->hwimman.defext to fnedit.defext.

If fnedit.flags contains both IN_FNEDIT_SET_DEFEXT and IN_FNEDIT_STANDARD, and DatUsedPathNamePtr points to a file specification, copies the extension from DatUsedPathNamePtr to the extension buffer fnedit.extbuf, and writes the address of the extension buffer to fnedit.defext. Clears IN_FNEDIT_SET_DEFEXT from fnedit.flags as the default extension is now set.

Otherwise if fnedit.flags contains just IN_FNEDIT_SET_DEFEXT, and par->fname specifies a filename followed by an extension, copies the file extension from par->fname to the extension buffer fnedit.extbuf, and writes the address of the extension buffer to fnedit.defext. Clears the IN_FNEDIT_SET_DEFEXT flag from fnedit.flags as the default extension is now set.

If fnedit.flags contains IN_FNEDIT_STANDARD, builds a full file specification from DatUsedPathNamePtr, using the p_fparse PLIB library function, and a NULL related file specification. Truncates the full file specification at the filename and write to fnedit.buf.

Otherwise, if fnedit.flags does not contain IN_FNEDIT_STANDARD, builds a full file specification from par->fname using the p_fparse PLIB library function, and a NULL related file specification. Truncates the full file specification at the filename and writes the result to fnedit.buf.

Sets the pack by sending a WN_SET message to fnedit.pack passing as arguments fnedit.buf and the address of the P_FPARSE struct written by the previous call to p_fparse.

Sets the text in the edit box by supersending a WN_INIT message passing as arguments a pointer to an init struct and landlord, the id of the landlord window. The init struct is defined as follows:

```
struct
    {
    IN_EDWIN ed;
    TEXT buf[P_FNAMESIZE];
    } init;
```

If fnedit.flags contains IN_FNEDIT_STANDARD, ed.contents[0] is set to zero.

If fnedit.flags contains the IN_FNEDIT_JUST_DIRS flag, par->fname is built into a full path specification, using the p_fparse PLIB library function, and a NULL related file specification. The full file specification is then truncated before the trailing directory and copied to ed.contents. (Thus "LOC::M:", "REM::C:\SIBOSDK\" and "REM::C\SIBOSDK\DEMO\" would become "LOC::M:", "REM::C:" and "REM::C:\SIBOSDK\" respectively.) Note that the IN_FNEDIT_JUST_DIRS flags is ignored if par->fname specifies a filename.

If neither the IN_FNEDIT_STANDARD nor the IN_FNEDIT_JUST_DIRS flags are set, the text specified by par->flags is built into a full file specification using the p_fparse PLIB library function and a NULL related file specification, and then copied to ed.contents.

Sets DLGBOX_ITEM_CAN_DEFER_X and DLGBOX_ITEM_X_PENDING in the dlgbox.item[].flags dialog property associated with this control, and returns.

WN_SET — Set file name

```
VOID wn_set(TEXT *fname);
```

Set the filename and, optionally, the default extension, according to the file specification pointed to by fname.

If the first character in fname.buf is '1', removes the first character and builds a full file specification from fname using the p_fparse PLIB library function, and a NULL related file specification. Writes the file extension to fnedit.extbuf, and truncates fname at the filename.

Builds a full file specification from fname using the p_fparse PLIB library function, and a NULL related file specification, and copies the result to fnedit.buf.

Sets the pack by sending a WN_SET message to fnedit.pack passing as arguments fnedit.buf, and the address of the P_FPARSE struct written by the previous call to p_fparse.

If fnedit.flags contains IN_FNEDIT_JUST_DIRS and fnedit.buf does not specify a filename, truncates fnedit.buf before the trailing directory. (Thus LOC::M:, REM::C:\TMP\ and REM::C\TMP\DEMO\ would become LOC::M:, REM::C: and REM::C:\TMP\ respectively.)

Truncates fnedit.buf at the filename if present.

Sets the full file specification into superclass property by supersending a WN_SET message.

WN_SENSE — Sense full file name

```
INT wn_sense(TEXT *fname);
```

Write to fname the full file specification of the current directory, or non-directory, file. Thus fname should be a pointer to a buffer of length at least P_FNAMESIZE.

Senses the file name by supersending a WN_SENSE message and copies the result to the buffer specified by fname. Builds a full file specification using the p_fparse PLIB library function, and a related specification of fnedit.buf. Writes the full file specification to fname. Returns the return value from the p_fparse function if it is non-zero.

If the full file specification contains one or more wildcards, and the IN_FNEDIT_CAN_WILDCARDS flag is set in fnedit.flags, returns -1 indicating legal use of wildcards.

If the full file specification contains one or more wildcards, and the IN_FNEDIT_CAN_WILDCARDS flag is not set in fnedit.flags, returns E_FILE_NAME indicating illegal use of wildcards.

If the full file specification contains a name, or an extension, and the IN_FNEDIT_JUST_DIRS flag is set in fnedit.flags, appends a delimiter character to the file specification if not already present. (Thus LOC::M:\TMP and REM::C:\TMP.DOS would become LOC::M:\TMP and REM::C:\TMP.DOS\ respectively.) Builds a full file specification using the p_fparse PLIB library function, with a NULL related file specification, and returns the return value.

If the full file specification contains a filename, and the IN_FNEDIT_JUST_DIRS flag is not set in fnedit.flags, uses the p_fparse PLIB library function to appends the default extension in fnedit.defext to fnedit.buf and returns the return value.

If the full file specification does not contain a name, and neither of the IN_FNEDIT_ACCEPT_NULL and IN_FNEDIT_ALLOW_DIRS flags are set in fnedit.flags, returns ERROR_RID_OFFSET-SYS_CHOOSE_FILENAME.

WN_KEY — Handle key input

```
INT wn_key(INT keycode,INT modifiers);
```

Handle the keypress specified by keycode and modifiers.

If a file selector is present and thus fnedit.pop is non-zero:

- allows the file selector to handle the keypress by sending a WN_KEY message to fnedit.pop with arguments of keycode and modifiers. If the user selects a valid file, sets this as the current file by sending self a WN_SET message. If the user selects a file that for some reason is not valid, *and* fnedit.flags does not contain IN_FNEDIT_JUST_DIRS, displays an appropriate information message and then returns WN_KEY_CHANGED.

- otherwise if the return value from the WN_KEY message is not WN_KEY_NO_CHANGE, destroys the pop-up file list by sending a DESTROY message to fnedit.pop and writing FALSE to fnedit.pop. Emphasises the file name editor by sending a WN_EMPHASISE message to self. Returns the return value from the WN_KEY message.

If keycode is a Tab, and modifiers contains the Psion modifier, displays the contents of fnedit.buf as an information message, and then returns WN_KEY_NO_CHANGE.

If keycode is a Tab and modifiers does not contain the Psion modifier, senses the full file specification for the current file by sending a WN_SENSE message to self:

- if modifiers contains the Control modifier, allows the user to edit the file name pattern and the full path for the current file by launching a *File list* dialog: if the user cancels the dialog, returns WN_KEY_NO_CHANGE.

- presents a list of the files in the parent directory of the current file by creating an instance of the FILELIST class, writing its handle to fnedit.pop, and sending a WN_INIT message to fnedit.pop.

- de-emphasises the file name editor by supersending a WN_EMPHASISE message.

- ensures that the control absorbs subsequent keypresses by returning WN_KEY_ABSORB_ON.

Otherwise lets the superclass handle the keypress by supersending a WN_KEY message, passing arguments of keycode and modifiers and then returns the return value.

WN_EMPHASISE — Emphasise

```
VOID wn_emphasise(INT flags);
```

Emphasise the control.

If fnedit.pop is non-zero, sends fnedit.pop a WN_EMPHASISE message passing as the argument flags. Otherwise supersends a WN_EMPHASISE message passing as the argument flags.

LG_SELF_CHECK — Validate file name

```
INT lg_self_check(VOID);
```

Validate the file name. On success, return TRUE. Otherwise, display an appropriate error message, using the hInfoPrintErr utility function, and return FALSE.

Sends a WN_SENSE message to self. If the return value is either E_FILE_NXIST, E_FILE_DIR or -1, returns TRUE. If the return value is non-zero, displays an appropriate error message and returns FALSE.

Otherwise performs further checks, which may lead to the hInfoPrintErr utility function displaying an error messsage corresponding to one of the following error codes:

E_FILE_NAME	invalid file name
E_FILE_DEVICE	invalid device name or the device does not exist
E_FILE_DIR	invalid directory name or the directory does not exist
E_GEN_FSYS	invalid file system name, or the file system does not exist
E_FILE_NOTREADY	the device does not contain a medium
E_FILE_NXIST	the file does not exist
E_FILE_RDONLY	the file is read-only
E_FILE_EXIST	the file is not a directory file and fnedit.flags contains IN_FNEDIT_FORCE_NXIST

ERROR_RID_OFFSET-SYS_CHOOSE_NEW_DIRECTORY	the file is a directory file and fnedit.flags contains IN_FNEDIT_FORCE_NXIST
ERROR_RID_OFFSET-SYS_CHOOSE_FILENAME	the file is a directory file and fnedit.flags does not contain IN_FNEDIT_ALLOW_DIRS
ERROR_RID_OFFSET-SYS_CHOOSE_DIRECTORY	the file is not a directory file and fnedit.flags contains IN_FNEDIT_JUST_DIRS

(The error codes are produced by the p_finfo and p_testpth PLIB library routines.)

If the file already exists, and fnedit.flags does not contain IN_FNEDIT_NO_AUTOQUERY, presents a query dialog with the SYS_EXISTS_OVER resource. This informs the user that the file exists and asks if it should be overwritten. If the user confirms, returns TRUE. Otherwise returns FALSE.

LG_UPDATE — Update with new path

```
VOID lg_update(TEXT *path);
```

Update the current path.

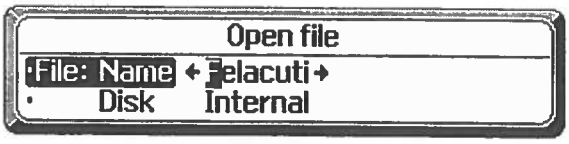
Builds a full file specification from path using the p_fparse PLIB library function, and a related file specification of fnedit.buf, and copies the result to fnedit.buf. Sets the DLGBOX_ITEM_X_PENDING flag in the dlgbox.item[i].flags property associated with the control.

FNSELWN

WIN	LODGER	CHLIST
flags id	landlord offset width	data flags matcher matchlen nsel pop
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ ~~wn_init~~ wn_visible lg_draw ~~lg_self_check~~ lg_set_id_pos ~~lg_sense_width~~ ~~lg_update~~	destroy wn_draw wn_emphasise ~~wn_init~~ ~~wn_key~~ ~~wn_sense~~ ~~wn_set~~ ~~lg_sense_width~~

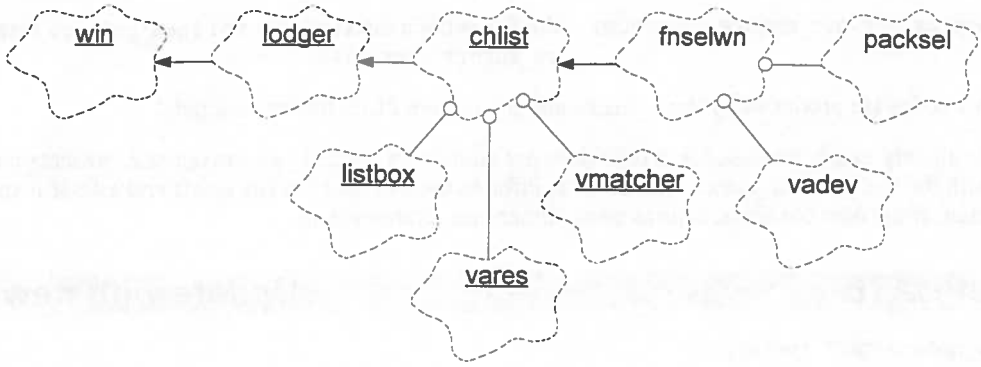
FNSELWN
tags pack flags defext extbuf crk buf
destroy wn_init wn_set wn_sense wn_key lg_self_check lg_sense_width lg_update fns_insert_tags fns_extract_tags fns_subset fns_validate_flist

The FNSELWN class implements the file name choice list control which allows the user to select an already existing file. An example file name choice list is shown in the following picture:



A file name choice list may have an associated pack selector control simply by ORing the DLGBOX_ITEM_NEEDS_PACK flag into the flags member of the FNSELWN resource. For details see the introduction to the FNEDIT control.

Class diagram



Class definition

Defined in sub-category file *files.cl* (generated header file *files.g*).

```
CLASS fnselwn   chlist
    {
    REPLACE destroy
    REPLACE wn_init
    REPLACE wn_set
    REPLACE wn_sense
    REPLACE wn_key
    REPLACE lg_self_check
    REPLACE lg_sense_width
    REPLACE lg_update
    ADD fns_insert_tags
    ADD fns_extract_tags
    ADD fns_subset=p_dummy
    ADD fns_validate_flist=p_true

    CONSTANTS
        {
        IN_FNSELWN_STANDARD             H_FILE_STANDARD_INIT
        IN_FNSELWN_SHOW_DIRS            H_FILE_ALLOW_DIRS
        IN_FNSELWN_HIDE_FILES           H_FILE_JUST_DIRS
        IN_FNSELWN_RESTRICT_LIST        H_FILE_RESTRICT_LIST
        IN_FNSELWN_CAN_TAG              H_FILE_CAN_TAG
        IN_FNSELWN_ACCEPT_NULL          H_FILE_ACCEPT_NULL
        IN_FNSELWN_SET_DEFEXT           H_FILE_SET_DEFEXT
        IN_FNSELWN_CAN_WILDCARD         H_FILE_CAN_WILDCARD
        PR_FNSELWN_TAGSHERE             0x100    Tags in this directory
        PR_FNSELWN_AT_ROOT              0x200
        PR_FNSELWN_WILDCARDED           0x400
        PR_FNSELWN_PRESERVE_TAGS        0x800
        PR_FNSELWN_ON_DIR               0x1000
        PR_FNSELWN_CHANGED_DIR          0x8000
        }

    TYPES
        {
        typedef struct
            {
            UBYTE flags;
            TEXT fname[1];
            } IN_FNSELWN;
        }
```

```
    PROPERTY
        {
        PR_VASTR *tags;          which files are tagged
        PR_PACKSEL *pack;
        UWORD flags;
        TEXT *defext;
        TEXT extbuf[6];
        P_FPARSE crk;            cracked information on current file
        TEXT buf[P_FNAMESIZE];
        }
    }
```

Property

`fnselwn.tags` this is either NULL or the handle of an instance of the VASTR class that is used to store references to tagged files.

If the instance of VASTR exists, its first record contains the full path of the directory containing the tagged files. Each subsequent record contains the name of a tagged file. Note that this means that all tagged files must be in the same directory.

`fnselwn.pack` this is either the handle of an instance of the PACKSEL class - used to select the pack - or NULL.

`fnselwn.flags` an ORed combination of flags that determine the behaviour of the FNSELWN instance.

`fnselwn.defext` this points to the default extension.

`fnselwn.extbuf` this is the default extension.

`fnselwn.crk` a P_FPARSE struct corresponding to the full file specification in `fnselwn.buf`.

`fnselwn.buf` the full file specification of the current file.

Tagged files

On pressing the Tab key when an instance of FNSELWN has focus, a file list is presented (using an instance of FILELIST). One or more of the file names in this list may be shown as being tagged, and tags may be set or cleared by using two language-dependent keypresses - on English machines, these are '+' and '-' (see the FILELIST `wn_key` method in the *FILELIST Window Class* chapter).

The current list of tagged files, if any exist, is maintained in an instance of the OLIB VASTR class whose handle is stored in `fnselwn.tags`. The first entry of this instance contains the full path name of the directory containing the tagged files and subsequent entries contain the names of the tagged files.

The VASTR instance may optionally be created by FNSELWIN on initialisation or it may be supplied externally, via the `fns_insert_tags` method. This method provides an option for the instance to remain under the ownership of the supplier. If this option is selected, the instance of VASTR will not be destroyed when FNSELWN receives a DESTROY message. If this option is not selected, or if the instance of VASTR has been created by FNSELWN, destruction of FNSELWN will also destroy the tagged files list..

FNSELWN methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the FNSELWN instance.

If `fnselwn.flags` does not contain PR_FNSELWN_PRESERVE_TAGS, and `fnselwn.tags` is non-zero, sends a DESTROY message to `fnselwn.tags`.

Supersends a DESTROY message.

WN_INIT — Initialise

```
VOID wn_init(IN_FNSELWN *par,PR_WIN *landlord,PR_PACKSEL *pack);
```

Initialise the file name choice list.

Writes `landlord`, the ID of the landlord window, to `lodger.landlord`, and sets `DLGBOX_ITEM_CAN_DEFER_X` and `DLGBOX_ITEM_X_PENDING` in the `dlgbox.item[i].flags` property associated with the control. Then writes `pack` to `fnselwn.pack` and sets `PR_CHLIST_PACKSEL_DODINFO` in `chlist.flags`.

If `fnselwn.flags` contains `IN_FNSELWN_HIDE_FILES`, sets `IN_FNSELWN_SHOW_DIRS` and clears `IN_FNSELWN_CAN_WILDCARD`, `IN_FNSELWN_CAN_TAG` and `IN_FNSELWN_SET_DEFEXT`.

Creates an instance of the `VASTR` class and writes its handle to `chlist.data`. Initialises the `VASTR` component by sending an `VA_INIT` message to `chlist.data` specifying a granularity of 16. Requests the `VASTR` component to sort the records alphabetically ignoring case by sending a `VA_KEY` messsage to `chlist.data`.

Creates an instance of the `VMATCHER` class and writes its handle to `chlist.matcher`. Initialises the `VMATCHER` component by sending an `IM_INIT` message to `chlist.matcher`. The length of the match string is stored in `chlist.matchlen`, the maximum length of the match string is 128 and the data is stored in `chlist.data`.

If `par->flags` contains `IN_FNSELWN_CAN_TAG`, creates an instance of the `VASTR` class, writing its handle to `fnselwn.tags`, and sends a `VA_INIT` message with a granularity of 32.

If `par->flags` contains `IN_FNSELWN_STANDARD`, the filename is taken from `DatUsedPathNamePtr`, otherwise the filename is taken from `&par->fname[0]`.

If `par->flags` contains `IN_FNSELWN_SET_DEFEXT`, a `NULL` file name is specified, and `w_ws->wserv.flags` does not contain `PR_WSERV_FROM_HWIF`, returns. In this case it is intended that the filename will be set by a subsequent `WN_SET` message.

Otherwise, sets the extension by writing `w_am->hwimman.defext` to `fnselwn.defext`.

Sets the filename by sending a `WN_SET` message to self.

Clears the `IN_FNSELWN_STANDARD` flag from `fnselwn.flags`.

WN_SET — Change directory and/or selection

```
VOID wn_set(TEXT *fname);
```

Set the file or non-file directory specified by `fname` into property.

If the first character in `fname` is '!', sets the default extension by copying the extension in `fname` to `fnselwn.extbuf` and returns.

Sets `PR_FNSELWN_CHANGED_DIR`, and clears `PR_FNSELWN_AT_ROOT` in `fnselwn.flags`.

Builds a full file specification from `fname` by calling the `p_fparse` PLIB library routine with a `NULL` related file specification. If the device specified in `fname` is invalid, substitutes the default device. Writes the full file specification to `fnselwn.buf`, and the associated `P_FPARSE` struct to `fnselwn.crk`.

If `IN_FNSELWN_SHOW_DIRS` is set, and neither the filename nor the extension is specified in `fname`, clears `IN_FNSELWN_SET_DEFEXT` in `fnselwn.flags`. If the file is in the root directory, sets `PR_FNSELWN_AT_ROOT` in `fnselwn.flags`. Otherwise removes the trailing delimiter.

If `fnselwn.flags` contains `IN_FNSELWN_SET_DEFEXT`, and the extension in `fnselwn.buf` is less than five characters long, sets the default extension by copying the extension in `fnselwn.buf` to the extension buffer `fnselwn.extbuf`, and writing the address of the extension buffer to `fnselwn.defext`. Indicates that the extension has been set by clearing `IN_FNSELWN_SET_DEFEXT` in `fnselwn.flags`.

Sets the current pack according to the content of `fnselwn.buf` and `fnselwn.crk` by sending a `WN_SET` message to `fnselwn.pack`.

The remaining step is to build a list of the files in the parent directory using the `BuildFileList` function. The second argument in the call is the return value from the earlier `WN_SET` message. The handle for the variable array is stored in `chlist.data`.

Building a file list

The BuildFileList function is declared as follows:

```
LOCAL_C BuildFileList(PR_FNSELWN *self, INT derr);
```

It is called by the wn_set, lg_update and fns_insert_tags methods.

Displays a scanning busy message using the SYS_SCANNING system resource and the hBusyPrint utility function.

Resets the list of files by sending a VA_RESET message to chlist.data, and sets the selection to item zero by supersending a WN_SET message. Clears PR_CHLIST_SUSPENDED in chlist.flags, and clears both PR_FNSELWN_TAGSHERE and PR_FNSELWN_WILDCARDED in fnselwn.flags.

If fnselwn.flags contains the IN_FNSELWN_RESTRICT_LIST flag, *and* fnselwn.buf contains *both* a filename and an extension, replaces the filename in fnselwn.buf with an asterisk character. Thus LOC::M:\DIR\FILENAME.EXT would become LOC::M:\DIR*.EXT.

If fnselwn.flags contains IN_FNSELWN_CAN_TAG, and *either* fnselwn.flags does not contain IN_FNSELWN_CAN_WILDCARD, or fnselwn.crk.flags is zero, *and* fnselwn.tags contains multiple files sharing the path specified in fnselwn.buf, sets PR_FNSELWN_TAGSHERE in fnselwn.flags, and PR_CHLIST_SUSPENDED in chlist.flags, adds a record containing the SYS_FILES_TAGGED system resource by sending a VA_APPEND message to chlist.data, sets the selection to item zero by supersending a WN_SET message, and returns.

If derr is non-zero but not E_GEN_NOMEMORY, adds a record containing an appropriate error message by sending a VA_APPEND message to chlist.data, sets the selection to item zero by supersending a WN_SET message, and returns.

If derr is E_GEN_NOMEMORY, calls p_leave, with an argument of E_GEN_NOMEMORY.

Otherwise attempts to open a channel to the directory in fnselwn.buf using the p_open PLIB library function and the P_FDIR mode.

If the return value is E_GEN_NOMEMORY, calls p_leave, with an argument of E_GEN_NOMEMORY.

Otherwise, if the return value is non-zero, adds a record containing an appropriate error message by sending a VA_APPEND message to chlist.data, sets the current selection to item zero by supersending a WN_SET message, and returns.

If fnselwn.flags contains PR_FNSELWN_AT_ROOT, sends a VA_APPEND message to chlist.data to add a record containing the delimiter character which for the Series 3 filing system would be "\".

If fnselwn.buf contains wildcards (thus fnselwn.crk.flags is non-zero) and IN_FNSELWN_CAN_WILDCARD is set in fnselwn.flags, adds a record containing the filename and extension in fnselwn.buf by sending a VA_APPEND message to chlist.data. Sets the selection to item zero by supersending a WN_SET message. Sets PR_FNSELWN_WILDCARDED in fnselwn.flags and returns.

Otherwise reads the files in the current directory, ignoring volume name directories, and any file whose name starts with a full stop character.

Directory files are considered only if IN_FNSELWN_ALLOW_DIRS is set in fnselwn.flags. For each directory file, appends a delimiter character (taken from fnselwn.buf), and creates a new record containing the filename.

Non-directory files are considered only if IN_FNSELWN_JUST_DIRS is clear in fnselwn.flags. For each non-directory file, removes the extension from the file name if and only if it matches that in fnselwn.defext, and creates a new record containing the filename.

The new records are added by sending a VA_APPEND message to chlist.data, followed by a VA_INSERT message if a duplicate filename already exists. (Duplicate filenames may occur when scanning remote filing systems).

To allow subclassers to add their own functionality the method sends a FNS_SUBSET message to self once the file list has been built. The default method does nothing. Subclassers would thus replace the fns_subset method with their own variant. They may wish, for example, to further restrict the file list according to additional application specific flags.

If the number of files in the file list is zero, adds a record containing the SYS_NO_FILES system resource and returns.

Locates the file list index of the current file by sending a VA_SEARCH message to chlist.data and then sets the current selection by supersending a WN_SET message.

If fnselwn.flags contains IN_FNSELWN_STANDARD, sets the selection to item zero, unless the current file matches the first record and the number of records is greater than one, in which case sets the selection to item one.

Otherwise sets the current selection to either the record that matches the current file, or to the first record if no such record exists.

WN_SENSE — Get file name

```
VOID wn_sense(TEXT *fname);
```

Write the filename to fname.

Senses the filename by calling the FnselwnSense function with an argument of FALSE.

The FnselwnSense function

The FnselwnSense function is declared as follows:

```
LOCAL_C FnselwnSense(PR_FNSELWN *self, INT internal);
```

It is called by the wn_sense and wn_key methods.

If chlist.flags contains PR_CHLIST_SUSPENDED, and fnselwn.flags contains any of PR_FNSELWN_WILDCARDED, PR_FNSELWN_TAGSHERE, or PR_FNSELWN_AT_ROOT, copies the contents of fnselwn.buf to fname and returns.

If chlist.flags contains PR_CHLIST_SUSPENDED, and internal is TRUE, copies the contents of fnselwn.buf to fname and returns.

Otherwise if chlist.flags contains PR_CHLIST_SUSPENDED, but none of the other conditions specified above are satisfied, sets the first element of fname to zero and returns.

Senses the current selection by sending a VA_PBUF message to chlist.data with an argument of chlist.nsel.

Builds a full file specification from the current selection using the p_fparse PLIB library routine and a related file specification of fnselwn.buf truncated at the filename. Writes the full file specification to fname.

If the current selection is not a directory file, checks the content of fname using the p_finfo PLIB library routine. If the return value is non-zero or the status member of the P_INFO struct is P_FADIR, writes the default extension in fnselwn.defext to fname.

WN_KEY — Handle key input

```
INT wn_key(INT keycode,INT modifiers);
```

Handle the keypress.

If a file list is present and thus chlist.pop is non-zero, sends a WN_KEY message to chlist.pop.

- if the return value from the WN_KEY message is greater than zero - thus the user has selected a file - clears PR_FNSELWN_AT_ROOT in fnselwn.flags.If the selected file is in the root directory, sets PR_FNSELWN_AT_ROOT in fnselwn.flags. Validates the file by sending a FNS_VALIDATE_FLIST message to self and if the file fails the validation, calls p_leave with an argument of RUN_ACTIVE_USED. Otherwise, clears IN_FNSELWN_RESTRICT_LIST from fnselwn.flags, and sets the file into property by sending self a WN_SET message. Returns WN_KEY_CHANGED.
- if the return value from the WN_KEY message is WN_KEY_NO_CHANGE, returns WN_KEY_NO_CHANGE.
- if the return value from the WN_KEY message is neither greater than zero, nor WN_KEY_NO_CHANGE, destroys the file list by sending a DESTROY message to chlist.pop, and then writing zero to chlist.pop. Returns the return value from the WN_KEY message.

If keycode is a tab character, and modifiers contains the Psion modifier, displays the contents of fnedit.buf using the hInfoPrint utility function, and returns WN_KEY_NO_CHANGE.

If keycode is a tab character and modifiers does not contain the Psion modifier, sets IN_FNSELWN_CAN_TAG, IN_FNSELWN_HIDE_FILES, IN_FNSELWN_SHOW_DIRS and IN_FNSELWN_CAN_WILDCARD in fnselwn.flags. Senses the current filename by calling the FnselwnSense function with an argument of TRUE (see the description of the wn_sense method for details), and, if it is not in the root directory, strips the terminating delimiter character, then:

- if modifiers contains the Control modifier, allows the user to edit the file name pattern and the full path for the current file by launching a *File list* dialog: if the user edits neither the filename pattern nor full path, returns WN_KEY_NO_CHANGE.
- if the last keypress included the Control modifier, validates the file specification by sending self an FNS_VALIDATE_FLIST message. If it fails the validation, calls p_leave with an argument of RUN_ACTIVE_USED. Otherwise, clears IN_FNSELWN_RESTRICT_LIST from fnselwn.flags and sets the edited filename into property by sending self a WN_SET message. Returns WN_KEY_CHANGED.
- presents a list of files that match the current full file specification by creating an instance of the FILELIST file selector class, writing its handle to chlist.pop and sending it a WN_INIT message. Returns WN_KEY_ABSORB_ON.

Otherwise clears PR_FNSELWN_CHANGED_DIR from fnselwn.flags and supersends a WN_KEY message with arguments of keycode and modifiers. Returns the return value.

LG_SELF_CHECK — Validate file name

```
INT lg_self_check(INT can_defer);
```

If either fnselwn.flags contains any of PR_FNSELWN_WILDCARDED, PR_FNSELWN_TAGSHERE, IN_FNSELWN_ACCEPT_NULL or PR_FNSELWN_AT_ROOT, or chlist.flags does not contain PR_CHLIST_SUSPENDED, return TRUE. Otherwise beep using the hBeep utility routine and return FALSE.

LG_SENSE_WIDTH — Sense width

```
INT lg_sense_width(VOID);
```

Return the pixel width of the control.

The width is calculated as that of fifteen maximum-width characters in the appropriate font, plus a cushion of two pixels.

LG_UPDATE — Update filename

```
VOID lg_update(TEXT *pack,INT derr);
```

Update fnselwn.buf and fnselwn.crk according to the device specification in pack.

Builds a full file specification from pack, using the p_fparse PLIB library routine and a related file specification of fnselwn.buf. Writes the full file specification and the associated P_FPARSE struct to fnselwn.buf and fnselwn.crk respectively.

Builds a list of the files in the parent directory of the current file using the BuildFileList function with an argument of derr (see the description of the wn_set method), and writes the handle of the file list to chlist.data.

Sets DLGBOX_ITEM_X_PENDING in the dlgbox.item[].flags dialog property associated with this control, and sets PR_FNSELWN_CHANGED_DIR in fnselwn.flags.

FNS_INSERT_TAGS — Insert tags

```
VOID fns_insert_tags(PR_VASTR *tags,INT preserve,TEXT *buf);
```

Insert the list of tagged files in the VASTR array whose handle is tags and, optionally, set the path and/or default extension as specified by buf.

Stores the new tags by sending a destroy message to fnselwn.tags, writing tags to fnselwn.tags and setting IN_FNSELWN_CAN_TAG in fnselwn.flags.

If preserve is non-zero, adds PR_FNSELWN_PRESERVE_TAGS to fnselwn.flags, recording the fact that the list is considered to remain under the ownership of the supplier.

If buf is non-zero, sets the path specified by buf as the current path by sending a WN_SET message to self.

Otherwise builds a list of the files in the current directory using the BuildFileList function with an argument of zero (see the description of the wn_set method), and writes the handle of the file list to chlist.data.

FNS_EXTRACT_TAGS — Extract tags

```
PR_VASTR *fns_extract_tags(VOID);
```

Return a pointer to the VASTR instance used to store the list of tagged files or FALSE if no valid list exists.

Determines the number of tagged files by sending a VA_COUNT message to fnselwn.tags. If no files are tagged, the method returns FALSE.

Otherwise determines the path for the tagged files - this is stored in the first record - and if this does not match the path component in fnselwn.buf, returns FALSE. Otherwise sets the return value to fnselwn.tags and sets fnselwn.tags to zero (if FNSELWN owned the list, zeroing fnselwn.tags effectively removes this ownership).

FNS_SUBSET — Non-standard work on file list

```
VOID fns_subset(VOID);
```

The supplied method does nothing.

It is intended that subclassers replace this method to provide additional application specific functionality when building a file list. Subclassers may, for example, wish to further restrict the filelist according to application specific flags in fnselwn.flags. For details see the description of the wn_set method.

FNS_VALIDATE_FLIST — Validate file list

```
INT fns_validate_flist(TEXT *pbuf);
```

The supplied method returns TRUE.

It is intended that subclassers replace this method to provide the desired application-specific functionality.

CHAPTER 13

FILE LIST GENERATOR CLASSES

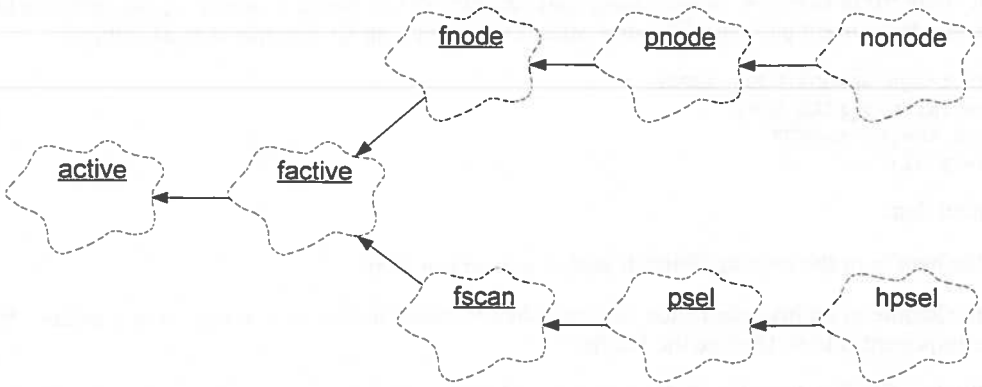
This chapter documents the NONODE and HPSEL classes which together support the creation and storage of a list of the files which match a specified path and filename pattern. The list of files may include file related details as required.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the file classes described in the *OLIB Reference* manual.
- the ACTIVE class described in the *OLIB Reference* manual.

Class diagram



NONODE

ACTIVE	FACTIVE	FNODE	PNODE
q priority isactive pcb stat	owner	flags pname oldname pcb	
destroy ~~ao_init~~ ~~ao_cancel~~ ~~ao_abrun~~ ~~ao_queue~~ ~~ao_run~~	~~ao_init~~ ao_cancel ~~ao_abrun~~ ~~fa_close~~	ao_queue ao_run fa_close fn_list ~~fn_end_list~~ ~~fs_nodename~~	ao_abrun fn_end_list fn_nodename

NONODE
fn_list

The NONODE class is provided for use by the HPSEL class and generates a list containg the default device and a list of files matching the current path and filename pattern.

Class definition

Defined in sub-category file *filelist.cl* (generated header file *filelist.g*).

```
CLASS nonode    pnode
    {
    REPLACE fn_list
    }
```

Property

None.

NONODE methods

FN_LIST — Initialise

```
VOID fn_list();
```

Create a node list containing the default device - e.g. `LOC::M:` - and start the creation of a list of files matching the current path and filename pattern.

Resets the node list by sending a `VA_RESET` message to `factive.owner->psel.pdir` and then clears both `PSEL_RESET_DIR_ARRAY` and `PSEL_QUEUED_CMD` in `factive.owner->psel.flags`.

Adds the default device to the node list by sending a `VA_APPEND` message to `factive.owner->psel.pdir`.

Closes the node list in `factive.owner->psel.pdir` and creates in `factive.owner->psel.pfile` a list of files matching the current path and filename pattern by sending an `AO_RUN` message as follows:

```
fnode.flags|=FNODE_NODE_ARRAY;
active.stat=E_FILE_EOF;
active.isactive=TRUE;
p_iosignal();
```

It is assumed that:

- the handle of the owning object is stored in `factive.owner`.
- the handle of an instance of the `PSELVAR` class is stored in `factive.owner->psel.pfile` - this component is used to store the file list.
- the handle of an instance of the `VASTR` class is stored in `factive.owner->psel.pdir` - this component is used to store the node list.
- the current path and filename pattern are stored as a zero terminated string in `factive.owner->psel.fspec`.

Returns zero.

HPSEL

ACTIVE	FACTIVE	FSCAN	PSEL	
q priority isactive pcb stat	owner	flags index pname match delim finfo crk pcbarr name	pfile pdir pnode dirent flags asccnt	dirnum setpath builderr fck fspec
destroy ~~ao_init~~ ~~ao_cancel~~ ~~ao_abrun~~ ~~ao_queue~~ ~~ao_run~~	~~ao_init~~ ~~ao_cancel~~ ~~ao_abrun~~ ~~fa_close~~	~~ao_queue~~ ao_run fa_close fs_matchname fs_fscan ~~*fs_fscan_end*~~ ~~*fs_dirname*~~ ~~*fs_filename*~~ *fs_end_dirlist*	~~ao_init~~ ao_abrun ao_cancel fs_filename fs_dirname fs_fscan_end ps_get_file ps_ascend_path ps_descend_path	ps_set_path ps_sense_filename ps_select_direntry ps_drives ps_settag ps_gettag ps_order ~~*ps_new_list*~~

HPSEL
owner
ao_init ao_queue fs_scan ps_new_list

The HPSEL class supports the creation of a list of files that match the current path and filename pattern. Note that the owning class must support the `fl_list_complete` method.

Class definition

Defined in sub-category file *filelist.cl* (generated header file *filelist.g*).

```
CLASS hpsel      psel
    {
    REPLACE ao_init
    REPLACE ao_queue
    REPLACE fs_fscan
    REPLACE ps_new_list
    PROPERTY
        {
        PR_ROOT *owner;
        }
    }
```

Property

`hpsel.owner` The handle of the owning object, assumed to be an instance of (a subclass of) `FILELIST`.

HPSEL methods

AO_INIT — Initialise

```
VOID ao_init(TEXT *fname,PR_ROOT *owner);
```

Initialise the HPSEL instance and create a list of files matching the possibly wildcarded path and filename pattern in `fname`.

Records the handle of the owning object by writing `owner` to `hpsel.owner`.

Supersends an `AO_INIT` message, to:

- write the path and filename pattern specified by `fname` to `psel.fspec`

- create an instance of the VASTR class having a granularity of 32 and write the handle to psel.pdir
- create an instance of the PSELVAR class having a granularity of 32 and write the handle to psel.pfile
- create an instance of the PNODE class and write the handle to psel.pnode., then initialise the PNODE instance by sending an AO_INIT message to psel.pnode.

Changes the class of which psel.pnode is an instance from PNODE to NONODE.

If an asynchronous request is outstanding, cancels the outstanding request and then adds a record holding the default device specification to psel.pdir and adds to psel.pfile a record for each file matching the path and filename pattern in psel.fspec. (The method sends AO_CANCEL and FN_LIST messages to psel.pnode.)

AO_QUEUE — Queue

```
VOID ao_queue(VOID);
```

Queue a read request.

If the request is to build the node list - i.e. fscan.flags contains FS_DIRECTORIES -

- clears both PSEL_RESET_DIR_ARRAY and PSEL_QUEUED_CMD in psel.flags.
- removes all records from the node list by sending a VA_RESET message to psel.pdir.
- adds the default device to the node list by sending a VA_APPEND message to psel.pdir.
- closes the device list and starts the creation of the file list using the following code:

```
active.stat=E_FILE_EOF;
active.isactive=TRUE;
p_iosignal();
```

  where the last line forces self to be sent an AO_RUN message.

Otherwise queues a request to add the next file to the file list by supersending an AO_QUEUE message.

FS_FSCAN — Start scan

```
VOID fs_fscan(TEXT *path);
```

Start the scan of the files and directories in the directory specified by path including hidden and system files - note that the file specification pointed to by path is overwritten

Sets FS_HIDDEN and FS_SYSTEM in fscan.flags and then starts the scan by supersending an FS_FSCAN message with an argument of path.

PS_NEW_LIST — Data rebuilt

```
VOID ps_new_list(VOID);
```

Complete the processing of the file list.

Allows the owning class to perform further processing of the completed file list by sending an FL_LIST_COMPLETE message to hpsel.owner.

CHAPTER 14

THE FILELIST WINDOW CLASS

This chapter documents the FILELIST class which may be used to create a list box containing a list of files and directories in the specified path.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

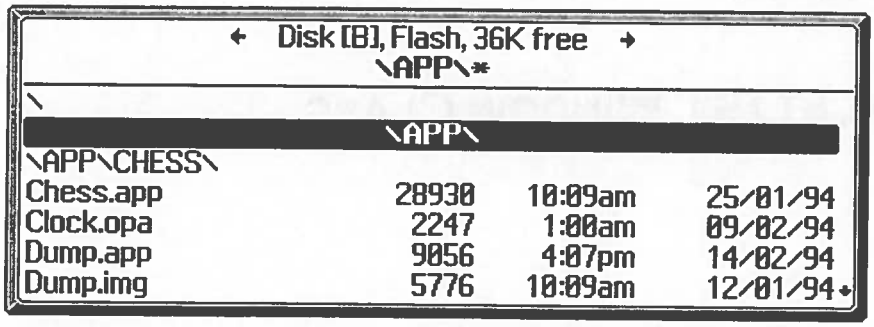
- the use of the file selector - obtained in the system screen by pressing the Tab key. The file selector is an instance of the FILELIST class.
- the VASTR class described in the *Variable Array Classes* chapter of the *OLIB Reference* manual.
- the VADEV class described in the *File Selectors* chapter of the *HWIM Reference* manual.

FILELIST

WIN	BWIN	LISTBOX	
flags id		match va flags width matchlen curoff offset	current top first last vastart matchstart
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help wn_visible *wn_set* ~~*wn_sense*~~ ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ ~~wn_emphasise~~	~~destroy~~ ~~wn_init~~ ~~wn_key~~ wn_draw wn_emphasise ~~lb_draw_item~~ ~~lb_draw_emphasis~~	lb_size_window lb_item_width lb_take_focus lb_inquire_focus ~~lb_inquire_item~~ lb_inquire_last

FILELIST	
hpsel contexts date tags ptags next locmask flags parent	thisdir crk derr x dfree wild subd scr targ
destroy wn_init wn_key wn_sense lb_draw_item	lb_draw_emphasis lb_enquire_item fl_list_complete fl_locchg

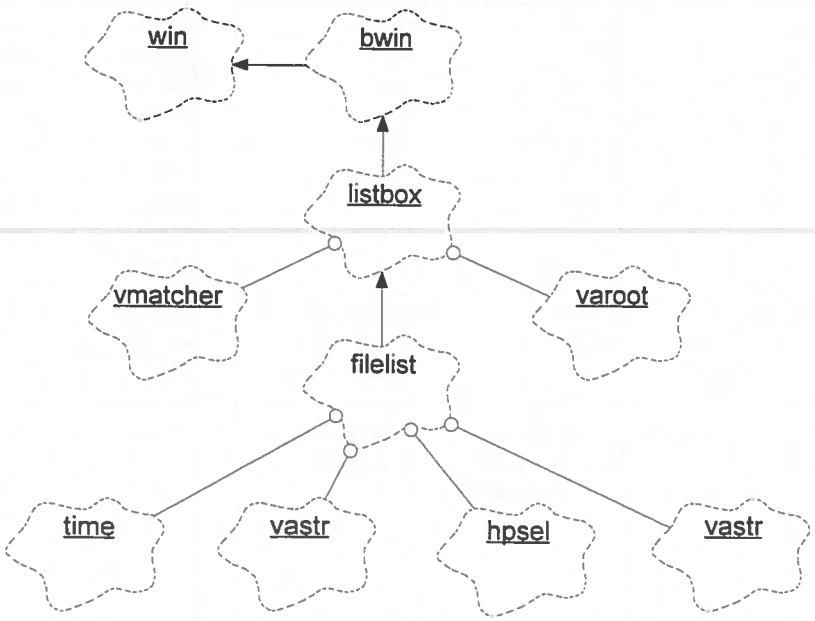
The FILELIST class implements the file list which presents the user with a list of the files in a given directory. The user may change the path using the cursor keys. The user may also select the node by pressing the appropriate letter e.g. 'B'. An example file list is shown in the following picture:



The file list displays:

- the name and type of the current device and the amount of free memory: these constitte the first title line.
- the current path and filename pattern which in the above example is *\APP*.
- the path of the parent directory: by selecting this line and pressing the Enter key the path ascends to the parent directory. In the above example this is \.
- the current path which in the above example is *\APP*.
- a list of the directories in the current path e.g. \APP\CHESS\ - by selecting a directory and pressing the Enter key the current path descends onto the selected directory.
- a list of the files in the current path and the associated file information. This consists of the a tag marker, the size of the file in bytes, the time at which the file was last modified and the date at which the file was last modified.

Class diagram



Class definition

Defined in the sub-category file *filelist.cl* (generated header file *filelist.g*).

```
CLASS filelist  listbox
    {
    REPLACE destroy             remove from list kept by wserv
    REPLACE wn_init             initialise listbox and do wn_set
    REPLACE wn_key              convert \ to special and filter ENTER and arrows
    REPLACE wn_sense            return current selection
    REPLACE lb_draw_item        draw all the file infomation
    REPLACE lb_draw_emphasis=pulldown_lb_draw_emphasis
    REPLACE lb_inquire_item     get pointer to text
    ADD fl_list_complete        called by fsel when file list is complete
    ADD fl_locchg               check for change in LOC::

    CONSTANTS
        {
        PR_FILELIST_ISFILE      0x100   Set or cleared as each line is drawn
        PR_FILELIST_TAGS_CHANGED        0x0200 Need to start a new tag list
        PR_FILELIST_NOSCAN              0x0400 No file scan for this directory
        PR_FILELIST_SCANNING            0x0800 Scan in progress
        PR_FILELIST_LOCAL               0x1000 Logged to LOC::
        PR_FILELIST_WILDCARDED          0x2000 Effects next wn_sense
        PR_FILELIST_SET_UNMARKED        0x4000 Called p_unmarka
        IN_FILELIST_ALLOW_DIRS          H_FILE_ALLOW_DIRS
        IN_FILELIST_JUST_DIRS           H_FILE_JUST_DIRS
        IN_FILELIST_CAN_TAG             H_FILE_CAN_TAG
        IN_FILELIST_CAN_WILDCARD        H_FILE_CAN_WILDCARD
        IN_FILELIST_NO_FORCE_WILD       0x8000

        FILENAME_WIDTH          12
        FILESIZE_WIDTH          7
        FILETIME_WIDTH          7
        FILEDATE_WIDTH          9
        }

    TYPES
        {
        typedef struct
            {
            WORD dtype;             disc type
            VOID *vadev;            device list
            WORD ascoff;            offset to where display should start
            UWORD tagoff;           x-offset to filetag
            UWORD sizeoff;          x-offset to filesize
            UWORD timeoff;          x-offset to filetime
            UWORD dateoff;          x-offset to filedate
            UWORD dateendoff;       x-offset to end of filedate
            TEXT asc[P_FNAMESIZE];  ascend text
            } PR_FILELIST_X;
        }
```

```
PROPERTY 4
    {
    PR_HPSEL *hpsel;          file list generator
    PR_VASTR *contexts;       contexts for all nodes encountered
    PR_TIME *date;            for file date display
    PR_VASTR *tags;           which files are tagged
    PR_VASTR **ptags;         where to write back tags
    PR_ROOT *next;            next filelist in list
    UWORD locmask;            mask for locchg
    UWORD flags;
    WORD parent;              index of parent (if any) in list
    WORD thisdir;             index of current directory in list
    P_FPARSE crk;             parse info on wildcard string
    WORD derr;                disc error (if any)
    PR_FILELIST_X *x;         extra data
    ULONG dfree;              size of free space on disc
    TEXT wild[P_FNAMESIZE]; holds wildcard string used by psel
    TEXT subd[P_FNAMESIZE]; used to generate subdir names
    TEXT scr[P_FNAMESIZE];  miscellaneous scratch buffer
    TEXT targ[P_FNAMESIZE]; where to position initial highlight
    }
}
```

Property

`filelist.hpsel` The handle of an instance of the HPSEL class. This is used to create and store a list of files and subdirectories. The full path and filename pattern that control the search for files and directories is in `filelist.hpsel->psel.fspec`. A full path and filename pattern of `LOC::M:\WVE\*` would list all files in the `LOC::M:\WVE\` directory.

`filelist.contexts` The handle of an instance of the VASTR class. There is one record for each node holding the most recent *scanning context* for that node i.e. the most recent full path and filename pattern.

`filelist.date` The handle of an instance of the TIME class. This is used to convert system time into textual representations of the time and date as required for the file list display.

`filelist.tags` The handle of an instance of the VASTR class. This is used to store details of the tagged files. The first record is used to store the full path. The remaining records hold the filenames. Thus the tagged files lie in the same directory as each other.

`filelist.ptags` A pointer to the handle of an instance of the VASTR class: this is for use when seeding the file list, in which case the VASTR array contains details of the currently tagged files. The user may supply the handle of this array on initialisation. Otherwise it is created by the `wn_sense` method if required.

`filelist.next` May be used to store the handle of the next FILELIST object in order to create a linked list of FILELIST objects.

`filelist.locmask` A mask that is for internal use only.

`filelist.flags` An ORed combination of flags indicating the state of the file list.

`filelist.parent` The index of the parent directory item. This is the first item in the main list unless the current directory is at the root. The user moves to the parent directory by highlighting this item and pressing the Enter key.

`filelist.thisdir` The index of the current directory item. This is the second item in the main list unless the current directory is at the root. In this case the item is not included.

`filelist.crk` This contains information about the length of the components and the presence of wildcards in the file specification pointed to by `filelist.wild`. The information is obtained by calling `p_fparse` with a file specification of `filelist.wild` and a NULL related file specification.

`filelist.derr` An error code indicating the status of the current device. The current device is in `filelist.wild`. The error code is generated using the `p_dinfo` routine.

`filelist.x` A pointer to a `PR_FILELIST_X` struct the definition of which is machine specific. The following members may be accessed:

- `vadev` the handle of an instance of `VADEV` which is used to store a list of the available devices e.g. `LOC::M:`, `LOC::A:` and `LOC::B:` - *may be read by a subclass.*
- `ascoff` the offset of the path component from `asc` - *may be read by a subclass.*
- `dtype` stores the ID of a system resource: this resource contains information about the media type in the current device - *may be read by a subclass.* The system resource ID corresponds to an error message or to one of the following:
  - `SYS_MTYPE_FLOPPY` specifies that the media is a floppy disk.
  - `SYS_MTYPE_HARD` specifies that the media is a hard disk.
  - `SYS_MTYPE_RAM` specifies that the media is a RAM disk.
  - `SYS_MTYPE_FLASH` specifies that the media is a flash disk.
  - `SYS_MTYPE_ROM` specifies that the media is a ROM disk and is thus read-only.
  - `SYS_MTYPE_PROTECTED` specifies that the media is read-only.
  - `SYS_MTYPE_UNKNOWN` specifies that the media type is unknown.
- `tagoff` the horizontal offset to the tag indicator - *for internal use only.*
- `sizeoff` the horizontal offset to the file size - *for internal use only.*
- `timeoff` the horizontal offset to the file time of last modification - *for internal use only.*
- `dateoff` the horizontal offset to the file date of last modification - *for internal use only.*
- `dateendoff` the horizontal offset to the end of the file date of last modification - *for internal use only.*
- `asc` points to the full path specification of the parent directory - stored as a zero terminated string - *for internal use only.*

`filelist.dfree` the amount of free space on the current device in units of Kb rounded to the nearest Kb. The current device is in `filelist.wild` - *for internal use only.*

`filelist.wild` the current path and filename pattern - stored as a zero terminated string - *for internal use only.* The current path and filename pattern for a file list showing all files in `LOC::M:\WVE\` would be `LOC::M:\WVE\*`, whereas the current path and filename pattern for a file list showing all files with extension `WVE` in `LOC::M:\WVE\` would be `LOC::M:\WVE\*.WVE`.

`filelist.subd` a format string used internally to generate the full file specification of a directory in the current directory - *for internal use only.* The format string for a file list showing files in `LOC::M:\WVE\` would be `LOC::M:\WVE\%S`.

`filelist.scr` a scrap buffer that is used internally - *for internal use only.*

`filelist.targ` see the description of the `wn_init` method for details - *for internal use only.*

FILELIST methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the filelist.

If `filelist.flags` contains `PR_FILELIST_SET_UNMARKED`, calls `p_marka`.

Sends a `WS_REMOVE_FILELIST` message to `w_ws` and then calls `wCancelBusyMsg`.

Supersends a `DESTROY` message.

WN_INIT — Initialise file list

```
VOID wn_init(INT flags,TEXT *fname,PR_VASTR **ptags);
```

Initialise the file list according to the content of `flags`, the file specification pointed to by `fname` and the handle of the `VASTR` array pointed to by `ptags`: the latter may be `NULL`.

The `flags` argument may contain an `ORed` combination of the following flags:

`IN_FILELIST_ALLOW_DIRS`	specifies that directories are inlcuded in the file list.
`IN_FILELIST_JUST_DIRS`	specifie that *only* directories are included in the file list.
`IN_FILELIST_CAN_TAG`	specifies that file tagging is allowed.
`IN_FILELIST_CAN_WILDCARD`	specifies that wildcards are allowed.
`IN_FILELIST_NO_FORCE_WILD`	specifies that the filename pattern is not automatically wildcarded i.e. it is not automatically replaced with an asterisk.

Writes `ptags` to `filelist.ptags` and writes `flags` to `filelist.flags`.

Defines the style of the file list window by setting `IN_BWIN_SHADOW_1`, `IN_BWIN_CUSHION` and `PR_WIN_EMPHASISED` in `win.flags`.

Connects to the window server by sending `self` a `WN_CONNECT` message specifying a width of `FILELIST_WIDTH` and a height of `FILELIST_HEIGHT` whilst ensuring that the file list is centred in the screen.

On the Work*about*, following connection to the window server, `ORS PR_LISTBOX_SMALL_FONT` into `win.flags`.

Creates an instance of the `VMATCHER` class - the incremental matcher class - and writes the handle to `listbox.match`. Initialises the `VMATCHER` component by sending an `IM_INIT` message to `listbox.match` passing as arguments the address of `listbox.matchlen` and `P_FNAMESIZE`.

Creates an instance of the `VASTR` class and writes the handle to `filelist.contexts`. Initialises the `VASTR` component by sending an `VA_INIT` message to `filelist.contexts` specifying a granularity of 32.

Creates an instance of the `TIME` class and writes the handle to `filelist.date`. Sets the format for the `TIME` instance (by sending `TO_SET_FORMAT` messages) so that the time is expressed in hours and minutes e.g. 14:32 or 2:32 pm and the date is expressed as the day, the month and the year e.g. 5/8/94.

Sets `PR_LISTBOX_KEEP_ARRAY` and `PR_LISTBOX_FORCE_WIDE` in `listbox.flags`.

Allocates a cell of sufficient size for a `PR_FILELIST_X` struct and writes the address of the cell to `filelist.x`.

Creates an instance of the `VADEV` class and writes the handle to `filelist.x->vadev`. Initialises the `VASTR` component by sending an `VA_INIT` message to `filelist.x->vadev`.

Builds a full file specification from `fname` using the `f_fparse` PLIB routine with a `NULL` related file specification, writes the result to `filelist.wild`, and writes the associated `P_FPARSE` struct to `filelist.crk`.

If filelist.wild contains wildcards, copies the current filename pattern in filelist.wild to filelist.targ and, if filelist.flags does not contain IN_FILELIST_NO_FORCE_WILD, replaces the filename pattern in filelist.wild with an asterisk.

Creates an instance of the HPSEL class and writes the handle to filelist.hpsel. Initialises the HPSEL component by sending an AO_INIT message to filelist.hpsel with arguments of filelist.wild and self.

Initialises items of LISTBOX property concerned with the display including the width which is FILELIST_INTERN_WIDTH and the cursor offset which is LISTBOX_LEFT_EDGE plus LISTBOX_OBLOID_INDENT.

If the current node in filelist.wild is LOC::, sets PR_FILELIST_LOCAL in filelist.flags.

Otherwise, clears PR_FILELIST_LOCAL in filelist.flags.

Writes an appropriate error code for the device specified by filelist.wild to filelist.derr - the error code is the return value from a call to p_dinfo thus zero corresponds to a useable device.

Writes the ID of an appropriate STRING resource giving information on the status of the device to filelist.x->dtype and, if the device is invalid, returns.

Writes the number of free bytes on the media, rounded to the nearest kilobyte, to filelist.dfree.

Sets PR_FILELIST_SCANNING in filelist.flags and displays the text in the SYS_SCANNING system resource using the wSetBusyMsg window server routine. On English language machines this is "Scanning".

Makes the file list visible by calling hInitvis, set PR_FILELIST_SET_UNMARKED in filelist.flags and calls p_unmarka.

WN_KEY — Handle key input

```
INT wn_key(INT keycode,INT modifiers);
```

Handle a keypress that may lead to a change in the current scanning context and hence file list.

If keycode is less than 27 and modifiers contains W_CTRL_MODIFIER:

- displays a scanning busy message.
- if keycode corresponds to a device - e.g. M or A or B, searches filelist.contexts for a record which matches the device and copies the path and filename pattern in the record to filelist.wild. (If there is no matching record, add a record containing the device specification to filelist.contexts.) If the device is LOC::, set PR_FILELIST_LOCAL in filelist.flags, otherwise, clears PR_FILELIST_LOCAL in filelist.flags.
- writes information about the status of the device to filelist.derr, filelist.x->dtype and filelist.dfree.
- creates a list of the files which match the path and filename pattern in filelist.wild as described in later paragraphs and returns WN_KEY_NO_CHANGE.

If filelist.flags contains either PR_FILELIST_NOSCAN or PR_FILELIST_SCANNING and the keypress is neither W_KEY_LEFT, nor W_KEY_RIGHT, nor W_KEY_TAB, nor W_KEY_ESCAPE, beeps and returns WN_KEY_NO_CHANGE.

If keycode is equal to w_ws->wserv.sc[H_SC_TPLUS] i.e. the special character that on English language machines is '+':

- if filelist.flags does not contain IN_FILELIST_CAN_TAG, or the current item is a directory, returns WN_KEY_NO_CHANGE.
- sets PR_FILELIST_TAGS_CHANGED in filelist.flags and tags the current item by sending a PS_SETTAG message to filelist.hpsel, redraws the item and returns WN_KEY_NO_CHANGE.

If keycode is equal to w_ws->wserv.sc[H_SC_TMINUS] i.e. the special character that on English language machines is '-':

- if filelist.flags does not contain IN_FILELIST_CAN_TAG, or the current item is a directory, returns WN_KEY_NO_CHANGE.

- sets PR_FILELIST_TAGS_CHANGED in filelist.flags, untags the current item by sending a PS_SETTAG message to filelist.hpsel, redraws the item and returns WN_KEY_NO_CHANGE.

If keycode is equal to w_ws->wserv.sc[H_SC_TSTAR] i.e. the special character that on English language machines is '*':

- if filelist.flags does not contain IN_FILELIST_CAN_TAG, returns WN_KEY_NO_CHANGE.
- sets PR_FILELIST_TAGS_CHANGED in filelist.flags and tags each file in the file list by sending PS_SETTAG messages to filelist.hpsel, redraws the display and returns WN_KEY_NO_CHANGE.

If keycode is equal to w_ws->wserv.sc[H_SC_TSLASH] i.e. the special character that on English language machines is '/':

- if filelist.flags does not contain IN_FILELIST_CAN_TAG, returns WN_KEY_NO_CHANGE.
- sets PR_FILELIST_TAGS_CHANGED in filelist.flags, untags each file in the file list by sending PS_SETTAG messages to filelist.hpsel, redraws the display and returns WN_KEY_NO_CHANGE.

If keycode is either W_KEY_UP or W_KEY_DOWN

- if modifiers contains W_SHIFT_MODIFIER, filelist.flags contains IN_FILELIST_CAN_TAG, and the current item is a file then sets PR_FILELIST_TAGS_CHANGED in filelist.flags. Toggles the tag status of the file by sending a PS_SETTAG message to filelist.hpsel. Redraws the file and file details.
- supersends a WN_KEY message with arguments of keycode and modifiers.
- cancels any scanning busy message and clears PR_FILELIST_SCANNING from filelist.flags.
- returns WN_KEY_NO_CHANGE.

If keycode is W_KEY_LEFT:

- searches filelist.contexts for a record which matches the current device in filelist.wild.
- moves to the *previous* record and copies the *scanning context* - i.e. path and filename pattern - to filelist.wild. (If there is no previous record, moves to the last record and repeats the process.)
- creates a list of the files which match the path and filename pattern in filelist.wild as described in later paragraphs.
- returns WN_KEY_NO_CHANGE.

If keycode is W_KEY_RIGHT:

- searches filelist.contexts for a record which matches the current device in filelist.wild.
- moves to the *next* record and copies the *scanning context* - i.e. path and filename pattern - to filelist.wild. (If there is no next record, moves to the first record and repeats the process.)
- creates a list of the files which match the current path and filename pattern in filelist.wild as described in later paragraphs.
- returns WN_KEY_NO_CHANGE.

If keycode is W_KEY_TAB:

- allows the user to edit the current path and filename pattern - stored in filelist.wild - by presenting a *File list* dialog : if the user modifies neither the full path nor the filename pattern, returns WN_KEY_NO_CHANGE.
- validates the full path in filelist.wild by removing trailing directories until the path exists.
- searches filelist.contexts for a record which matches the current device in filelist.wild and copies the current *scanning context* - i.e. current path and filename pattern - to the record. (If there is no matching record, appends a record containing the path and filename pattern in filelist.wild.)
- if on exiting the file list dialog the Control modifier was pressed, and filelist.flags contains IN_FILELIST_CAN_WILDCARD, sets PR_FILELIST_WILDCARDED in filelist.flags and returns

WN_KEY_CHANGED. (The last key press is stored in w_ws->wserv.ws.u.key.keycode and the modifiers are in w_ws->wserv.ws.u.key.modifiers.)

- displays a scanning busy message and if the current path is on LOC::, sets PR_FILELIST_LOCAL in filelist.flags, otherwise clears PR_FILELIST_LOCAL in filelist.flags.
- writes information about the current device in filelist.wild to filelist.derr, filelist.x->dtype and filelist.dfree.
- creates a list of the files which match the current path and filename pattern in filelist.wild as described in later paragraphs.
- returns WN_KEY_NO_CHANGE.

If keycode is W_KEY_RETURN:

- if the current item is the parent directory, displays a scanning busy message, and writes the full path specification of the parent directory followed by the current filename pattern to filelist.wild. Creates a list of the files which match the current path and filename pattern in filelist.wild as described in later paragraphs. Returns WN_KEY_NO_CHANGE.
- if the current item is a directory file, modifiers contains W_PSION_MODIFIER and filelist.flags contains IN_FILELIST_ALLOW_DIRS, returns WN_KEY_CHANGED.
- if the current item is the current directory, and filelist.flags contains IN_FILELIST_ALLOW_DIRS, returns WN_KEY_NO_CHANGE.
- if the current item is the current directory, and filelist.flags does not contain IN_FILELIST_ALLOW_DIRS, calls hInfoPrint with an argument of SYS_CHOOSE_FILENAME and returns WN_KEY_NO_CHANGE.
- if the current item is a file, and filelist.flags contains IN_FILELIST_JUST_DIRS, calls hInfoPrint with an argument of SYS_CHOOSE_DIRECTORY. Writes zero to listbox.matchlen and moves the focus to the current directory by sending self a LB_TAKE_FOCUS message. Returns WN_KEY_NO_CHANGE.
- writes the current path and filename pattern to filelist.wild.
- displays a scanning busy message and creates a list of the files which match the current path and filename pattern in filelist.wild as described in later paragraphs.
- returns WN_KEY_NO_CHANGE.

If keycode is none of the above:

- if modifers contains W_SHIFT_MODIFIER and keycode is an alphanumeric character, searches the file list for a directory the first character in the name of which matches keycode and moves the focus to the matching directory by sending self an LB_TAKE_FOCUS message. If no matching directory is found, beeps. In either case returns WN_KEY_NO_CHANGE.
- supersends a WN_KEY message with arguments of keycode and modifiers. If the return value is non-zero indicating that the display has changed, clears PR_FILELIST_SCANNING in filelist.flags and cancels any scanning busy message.
- returns WN_KEY_NO_CHANGE.

Creating a list of files matching the current path and filename pattern

If filelist.flags contains PR_FILELIST_TAGS_CHANGED and filelist.flags contains IN_FILELIST_CAN_TAG:

- clears PR_FILELIST_TAGS_CHANGED in filelist.flags.
- ensures that filelist.tags contains the handle of an instance of the VASTR class containing zero records of granularity 32.
- adds the current path and filename pattern in filelist.hpsel->psel.fspec to the tags array by sending a VA_APPEND message to filelist.tags.

- adds the name of each tagged file to the tags array by sending VA_APPEND messages to filelist.tags.

Writes information about the current path and filename pattern in filelist.wild to filelist.crk and writes zero to filelist.targ.

If the current device is not useable - thus filelist.derr is non-zero:

- calls hInfoPrint with an argument of filelist.x->dtype.
- sets PR_FILELIST_NOSCAN and clear PR_FILELIST_SCANNING in filelist.flags and cancels any scanning busy message.
- cancels any outstanding read request by sending an AO_CANCEL message to filelist.hpsel.

Otherwise creates a list of files and directories as follows:

- cancels any outstanding information message and displays a scanning busy message.
- set PR_FILELIST_SCANNING and clear PR_FILELIST_NOSCAN in filelist.flags.
- generates a list of files matching the full path and filename pattern in filelist.wild by sending a PS_SET_PATH message to filelist.hpsel.

Sets PR_LISTBOX_UNSTABLE in listbox.flags.

Resets the indices and redraws the file list.

Returns WN_KEY_NO_CHANGE.

WN_SENSE — Return current selection

```
INT wn_sense(TEXT *buf);
```

Sense the current item by writing a full file specification for the current item to buf.

If filelist.flags contains either PR_FILELIST_NOSCAN or PR_FILELIST_SCANNING, beeps and calls p_leave with an argument of RUN_ACTIVE_USED.

If filelist.flags contains IN_FILELIST_CAN_TAG:

- ensures that filelist.ptags points to the handle of an instance of the VASTR class containing zero records of granularity 32.
- adds the current path in filelist.hpsel->psel.fspec to the tags array by sending a VA_APPEND message to the array whose handle is pointed to by filelist.ptags.
- adds the name of each tagged file to the tags array by sending a VA_APPEND message to the array whose handle is pointed to by filelist.ptags.

If filelist.flags contains PR_FILELIST_WILDCARDED, copies the current path and filename pattern in filelist.wild to buf. Returns zero.

If the current item is neither the parent directory (i.e. listbox.current is not filelist.parent) nor the current directory (i.e. listbox.current is not filelist.thisdir) writes the full file specification for the current item to buf. If the file is a directory file, returns H_FILE_IS_DIR, otherwise returns zero.

If the current item is the current directory (i.e. listbox.current is equal to filelist.parent) copies the current path in filelist.wild to buf.

If the current item is the parent directory (i.e. listbox.current is equal to filelist.thisdir) copies the full path of the parent directory in filelist.x->asc to buf.

If the path in buf is at the root directory, returns H_FILE_IS_ROOT. Otherwise returns H_FILE_IS_DIR.

LB_DRAW_ITEM — Draw an item

```
VOID lb_draw_item(TEXT *txt,INT index,P_RECT *parea);
```

Draw the item specified by txt and index in the rectangle specified by parea.

If index is zero draws the zero terminated string specified by txt in a box specified by *parea. The text is drawn in normal style with centre alignment. The text should be the title. Returns.

If index is one draws the zero terminated string specified by txt in the box specified by *parea. The text is drawn in a bold style with centre alignment. The text should be the current path and filename pattern e.g. \WVE*. Returns.

If index is filelist.thisdir draws the zero terminated string specified by txt in the box specified by *parea. The text is drawn in a bold style with centre alignment. The text should be the name of the current directory e.g. \WVE\. Returns.

If index is filelist.parent draws the zero terminated string specified by txt in the box specified by parea. The text is drawn in a normal style with left alignment. The text should be the path for the parent directory e.g. "\". Returns.

Otherwise draws the zero terminated string specified by txt in the box specified by parea. The text is drawn in a normal style with left alignment. The text should be the file/directory name.

If filelist.flags contains PR_FILELIST_ISFILE the txt argument should point to the name member of a PSEL_REC struct.

The PSEL_REC struct is defined as follows:

```
typedef struct
    {
    UWORD flags;
    UWORD namlen;
    P_INFO info;
    UBYTE name[P_FNAMESIZE];
    } PSEL_REC;
```

The significance of the members of the PSEL_REC struct is as follows:

flags	contains PSEL_FLAG_TAG to indicate that the file is tagged, and zero otherwise.
namlen	the offset of the file extension in name.
info	additional file information including the time and date of last modification: see the *PLIB Reference* manual for details of the P_INFO struct.
name	the filename stored as a zero terminated string.

Draws additional file information to the right of the filename - this consists of a tag symbol if required, the size of the file in bytes, and the time and date of last modification.

LB_INQUIRE_ITEM — Get item text

```
TEXT *lb_inquire_item(INT index);
```

Return a pointer to the text of the item specified by index where the title has index zero.

Clears PR_FILELIST_ISFILE from filelist.flags.

If index is zero, returns a pointer to the title.

If index is equal to one, returns a pointer to the current path and filename pattern - e.g. \APP*. Note the absence of the device specification.

If index is equal to filelist.parent, returns a pointer to the path of the parent directory e.g. \. Note the absence of the device specification.

If index is greater than or equal to self->listbox.vastart and the item is a file sets PR_FILELIST_ISFILE in filelist.flags and returns a pointer to the name of the file e.g. "Chess.app". Note that the name forms part of a PSEL_REC struct as described in the description of the lb_draw_item method.

If index is greater than or equal to self->listbox.vastart and the item is a directory returns a pointer to the name of the directory e.g. LOC::M:\APP\CHESS.

FL_LIST_COMPLETE — List has been generated

```
VOID fl_list_complete(VOID);
```

Complete the processing of the file list generated by the `filelist.hpsel` component.

Cancels any scanning busy message and clears `PR_FILELIST_SCANNING` in `filelist.flags`.

If the paths in `filelist.wild` and `filelist.hpsel->psel.fspec` do not match - indicating that the file list has been reset for some reason:

- calls `hInfoPrint` with an argument of `SYS_FLIST_RESET`.
- copies the content of `filelist.hpsel->psel.fspec` to `filelist.wild`.
- if the current node is `LOC::`, sets `PR_FILELIST_LOCAL` in `filelist.flags`, otherwise clears `PR_FILELIST_LOCAL` in `filelist.flags`.
- writes information about the status of the device to `filelist.derr`, `filelist.x->dtype` and `filelist.dfree`.

Ensures that the file list displays the list of file names generated by the `HPSEL` component by writing `filelist.hpsel->psel.pfile` to `listbox.va`.

Enables incremental matching to the list of file names by writing `filelist.hpsel->psel.pfile` to `listbox.match.vmatcher.va` and sending an `IM_SET_RANGE` message to `listbox.match`.

If the current path and filename pattern in `filelist.hpsel->psel.pfile` is the `NULL` string indicating that the machine is out of memory:

- sets `PR_FILELIST_NOSCAN` in `filelist.flags`.
- clears the file list by writing appropriate values to property - see the *Property* section for details - then redraws the display and returns.

If the current directory is at the root e.g. `REM::D:`, writes zero to `filelist.parent` - since there is no parent directory - and writes two to `filelist.thisdir`.

Writes appropriate values to the remaining items of property - see the *Property* section for details.

If the name of a file in the current directory matches the zero terminated string in `filelist.targ`, sets the focus to that file.

Otherwise if `filelist.flags` contains `H_FILE_JUST_DIRS`, sets the focus to the current directory by sending `self` an `LB_TAKE_FOCUS` message.

Otherwise sets the focus to the first file in the file list by sending `self` an `LB_TAKE_FOCUS` message. If there are no files then sets the focus to the last directory. If the focus is now at the last item, moves the focus to the preceding item.

If `filelist.flags` contains `IN_FILELIST_CAN_TAG` and the path stored in the tags array matches the path in `filelist.wild::`

- for each tagged file sets `PSEL_FLAG_TAG` in the `flags` member of the `PSEL_REC` struct stored in `filelist.hpsel`.

Note that the method redraws the file list before sending an `LB_TAKE_FOCUS` message.

FL_LOCCHG — Check for change in LOC::

```
VOID fl_locchg(VOID);
```

Check for any changes on `LOC::`.

If `filelist.flags` contains `PR_FILELIST_SCANNING` or `filelist.flags` does not contain `PR_FILELIST_LOCAL`, return.

Determines whether the `LOC::` device has changed by calling `p_locchg` and if it has not changed, returns.

If the current node in `filelist.wild` is `LOC::`, sets `PR_FILELIST_LOCAL` in `filelist.flags`. Otherwise clears `PR_FILELIST_LOCAL` in `filelist.flags`.

Writes information about the status of the device to `filelist.derr`, `filelist.x->dtype` and `filelist.dfree`.

Sets `PR_LISTBOX_UNSTABLE` in `listbox.flags`.

If `filelist.derr` is non-zero - thus the current device is not useable:

- moves the current path in `filelist.wild` to the root - e.g. `LOC::M:*`.
- calls `hInfoPrint` with an argument of `filelist.x->dtype`.
- sets `PR_FILELIST_NOSCAN` and clear `PR_FILELIST_SCANNING` in `filelist.flags`.
- cancels any scanning busy message `filelist.flags`.
- cancels any outstanding read request by sending an `AO_CANCEL` message to `filelist.hpsel`.

Otherwise creates a list of files matching the path and filename pattern in `filelist.wild` as follows:

- writes the current device specification to `filelist.wild` and to the appropriate record in the `filelist.contexts` array.
- cancels any outstanding information message and display a scanning busy message.
- sets `PR_FILELIST_SCANNING` and clear `PR_FILELIST_NOSCAN` in `filelist.flags`.
- sends a `PS_SET_PATH` message to `filelist.hpsel`.

Sets `PR_LISTBOX_UNSTABLE` in `listbox.flags`.

Resets the display and redraws the file list.

CHAPTER 15

GENERAL SYSTEM DIALOGS

This chapter describes a number of dialog classes that are supplied, ready for use, by HWIM. The classes are as follows:

- the ERRORDLG class which provides a dialog containing an error message.
- the QUERYDLG class which provides a dialog containing one or two lines of text and an action list prompting the user for a Yes/No response.
- the FLISTDLG class which provides a dialog containing an editable path and filename pattern.
- the FONTSEL class which provides a dialog that allows the user to select the required font and font style.
- the EVALDLG class which provides a dialog that allows the user to select the system *Calculator* and *Evaluate* settings.
- the SETPORTDLG class which provides a dialog that allows the user to select the serial port settings.
- the SETHSHKDLG class which provides a dialog that allows the user to select the serial port handshake settings.

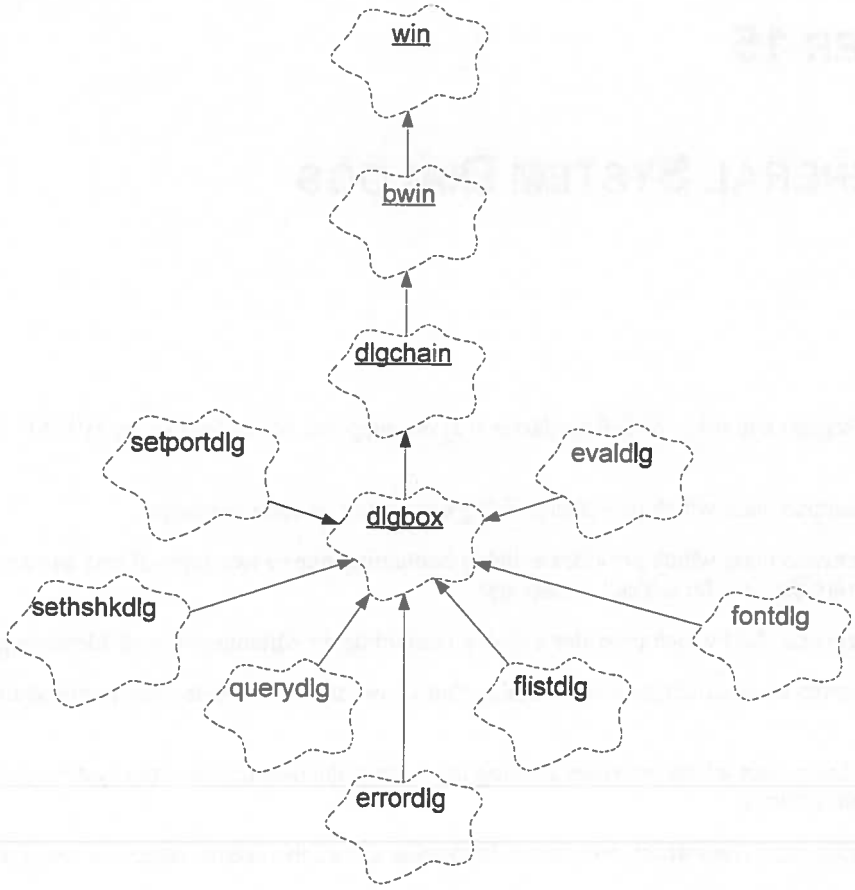
For details of the dialling dialogs see the *Dialling Dialogs* chapter of the *HWIM Reference* manual.

Precursors

Familiarity with the following topics would aid understanding of this chapter:

- the DLGBOX class which is described in the *Dialog Boxes* chapter of the *HWIM Reference* manual.
- the WDR class which is described in the *Printing* chapter of the *FORM Reference* manual
- *WDR* files which are described in the *WDR Printing* chapter of the *Additional System Information* manual

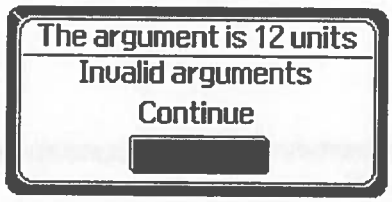
Class diagram



ERRORDLG

WIN	BWIN	DLGCHAIN	DLGBOX		ERRORDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ dl_key *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init

The ERORDLG class implements the *Error* dialog which presents the user with two lines of text followed by an action list containing a *Continue* button. An example *Error* dialog is shown in the following picture:



The simplest method of launching an *Error* dialog is to use the `ws_error_dialog` method of the `WSERV` object as described in the *WSERV Class* chapter of the *HWIM Reference* manual.

Class definition

Defined in sub-category file *dlgbox.cl* (generated header file *dlgbox.g*).

```
CLASS errordlg dlgbox
    {
    REPLACE dl_dyn_init
    TYPES
        {
        typedef struct
            {
            WORD error;
            TEXT message[50];
            } RBUF_ERROR;
        }
    }
```

Resources

Defined in system resource file `s_.rss`.

```
RESOURCE DIALOG sys_error_dialog
    {
    flags=DLGBOX_RBUF_FILLED|DLGBOX_NO_DDP;
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD|DLGBOX_ITEM_UNDERLINED;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_ac_continue;
                };
            }
        };
    }
```

Property

None.

ERRORDLG methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Initialise the items in the error dialog. It is assumed that `dlgbox.rbuf` contains the address of an `RBUF_ERROR` struct.

Writes `-1` to `dlgbox.helprid` indicating that there is no associated help resource.

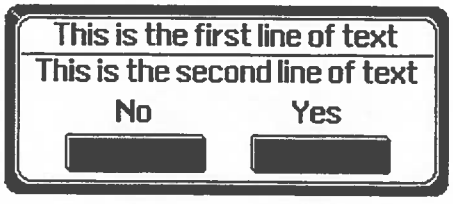
Sets the `TEXTWIN` control with index zero to display the text message specified in `dlgbox.rbuf->message`. The control is set by sending `self` a `WN_SET` message.

Sets the `TEXTWIN` control with index one to display the error message that corresponds to the error code in `dlgbox.rbuf->error`. The error code is converted to an error message using the `hErrs` utility routine. The control is set by sending `self` a `WN_SET` message.

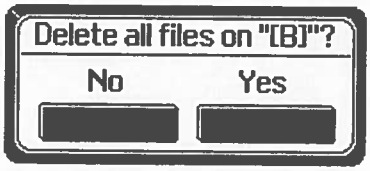
QUERYDLG

WIN	BWIN	DLGCHAIN	DLGBOX		QUERYDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~*wn_set*~~ ~~*wn_sense*~~ ~~*wn_draw*~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message ~~dl_item_add~~ dl_set_size dl_inq_minsize ~~dl_dyn_init~~ dl_key *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_item_add dl_dyn_init

The `QUERYDLG` class implements the *Query* dialog which presents the user with two lines of text followed by an action list containing *Yes* and *No* buttons. An example *Query* dialog is shown in the following picture:



A query dialog as its name suggest is most often used to request user confirmation as in the following example from the system screen:



The simplest method of launching a *Query* dialog is to use the `ws_query_dialog` method of the `WSERV` object as described in the *WSERV Class* chapter of the *HWIM Reference* manual.

Class definition

Defined in sub-category file *dlgbox.cl* (generated header file *dlgbox.g*).

```
CLASS querydlg dlgbox
    {
    REPLACE dl_item_add
    REPLACE dl_dyn_init
    TYPES
        {
        typedef struct
            {
            WORD result;
            TEXT message[50];
            } RBUF_QUERY;
        }
    }
```

Property

None.

Resources

Defined in system resource file *s_.rss*.

```
RESOURCE DIALOG sys_query_dialog
    {
    flags = DLGBOX_ACTION_LIST|DLGBOX_REPORT_ACT_HORIZ;
    controls=
        {
        CONTROL
            {
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD|DLGBOX_ITEM_UNDERLINED;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_ac_no_yes;
                };
            }
        };
    }

RESOURCE CONTROL sys_txtmess
    {
    class=C_TEXTWIN;
    flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD;
    info=TXTMESS
        {
        flags=IN_TEXTWIN_AL_CENTRE;
        };
    }
```

QUERYDLG methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Initialise the items in the dialog. It is assumed that `dlgbox.rbuf` contains the address of an `RBUF_QUERY` struct.

Sets the `TEXTWIN` control with index zero to display the text message specified by `dlgbox.rbuf->message`. The control is set by sending `self` a `WN_SET` message.

Sets the `TEXTWIN` control with index one to display the text in the resource whose id is `dlgbox.rbuf->result`: no text is displayed if the ID is zero. The control is set by sending `self` a `WN_SET` message.

Writes zero to `dlgbox.rbuf->result`.

DL_ITEM_ADD — Add item to end of list

```
VOID dl_item_add(AD_DLGBOX *par);
```

Add `TEXTWIN` controls for the text message and the resource text if specified.

Adds the dialog item specified by `par` by supersending a `DL_ITEM_ADD` message with `par` as argument.

If `dlgbox.count` is unity and `dlgbuf.rbuf->result` is non-zero, appends the `TEXTWIN` control specified by the `SYS_TXTMESS` system resource. The control is appended by sending `self` a `DL_ITEM_APPEND` message.

FLISTDLG

WIN	BWIN	DLGCHAIN	DLGBOX		FLISTDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_key dl_dyn_init

The `FLISTDLG` class implements the *File list* dialog which presents the user with an editable file name pattern and an editable full path. An example *File list* dialog is illustrated in the following picture:



An example of a *File list* dialog may be obtained while the file selector is present by pressing the Tab key.

Class definition

Defined in sub-category file *dlgbox.cl* (generated header file *dlgbox.g*).

```
CLASS flistdlg dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

Property

None.

Resources

Defined in system resource file *s_.rss*.

```
RESOURCE DIALOG sys_flist_dialog
    {
    title="Specify filelist";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_EDWIN;
            prompt="Filename pattern";
            info=EDWIN
                {
                maxlen=P_FNAMESIZE;
                vulen=18;
                flags=IN_EDWIN_VULEN_CHARACTERS;
                };
            },
        CONTROL
            {
            class=C_EDWIN;
            prompt="Full path";
            info=EDWIN
                {
                maxlen=P_FNAMESIZE;
                vulen=18;
                flags=IN_EDWIN_VULEN_CHARACTERS;
                };
            }
        };
    }
```

FLISTDLG methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Initialise the items in the dialog.

Builds a full file specification from the file specification pointed to by `dlgbox.rbuf`, using the `f_fparse` PLIB library routine, and a `NULL` related file specification.

If neither the filename, nor the extension, in the full file specification are wild carded, removes both the filename and the extension and appends a zero terminated asterisk.

Displays the filename component of the full file specification in the *Filename pattern* control.

Displays the full path components of the full file specification in the *Full path* control.

DL_KEY — Handle key input

```
INT dl_key(INT id,INT keycode);
```

Handle a key press.

Builds a full file specification from the text displayed in the control with index `id`, using the `p_fparse` PLIB library routine. The related file specification is the text displayed in the `EDWIN` control with index `3-id`.

If one or more components of the full file specification are invalid, displays an appropriate error message using the `hInfoPrintErr` utility function and returns `WN_KEY_NO_CHANGE`.

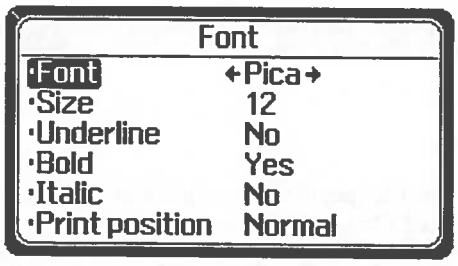
If the full file specification does not contain wild cards, and both filename and extension are absent, appends a zero terminated asterisk.

Writes the full file specification to the address in `dlgbox.rbuf` and returns `WN_KEY_CHANGED`.

FONTSEL

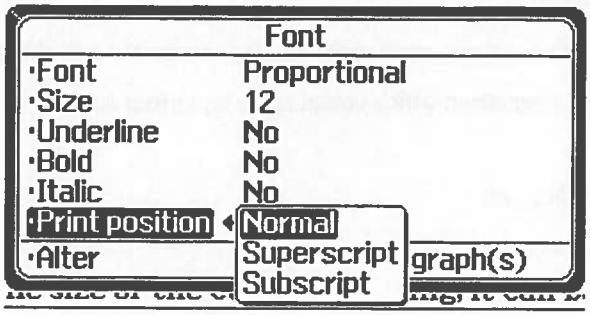
WIN	BWIN	DLGCHAIN	DLGBOX		FONTSEL
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	names sizes wdr pf f
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~*wn_set*~~ ~~*wn_sense*~~ ~~*wn_draw*~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ ~~dl_changed~~ *dl_focus* *dl_launch_sub* *dl_item_new*	dl_key dl_changed dl_dyn_init

The `FONTSEL` class implements the *Font selector* dialog which presents the user with selectable font, font size, font style and print position items. An example *Font selector* dialog is shown in the following picture:



(Note that the *Font* control displays typefaces and *not* fonts, although the meaning is clear.)

Another example of a font selector dialog is provided by the Word application as illustrated in the following picture (the dialog is obtained by selecting the *Font* item from the *Paragraphs* menu):



The dialog also illustrates the pop-up menu for the *Print position* item: the user may select *Normal*, *Superscript* or *Subscript*. Notice the extra item that has been appended to the dialog.

The `dlgbox.rbuf` property *must* point to a `FONTSEL_DATA` struct defined as follows:

```
typedef struct
    {
    VOID *wdr;
    SCRLAY_FONT *pf;
    UWORD ret;
    } FONTSEL_DATA;
```

The significance of the members of the `FONTSEL_DATA` struct is as follows:

`wdr` specifies the handle of the current `WDR` object.

`pf` specifies the address of a `SCRLAY_FONT` struct containing the font data to be edited.

`ret` the return value set by the `dl_key` method - it is set to a non-zero value if any of the font characteristics have been edited and to zero otherwise.

Class definition

Defined in sub-category file *fontdlg.cl* (generated header file *fontdlg.g*).

```
CLASS fontsel   dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_changed
    REPLACE dl_key
    TYPES
        {
        typedef struct
            {
            VOID *wdr;
            SCRLAY_FONT *pf;
            UWORD ret;
            } FONTSEL_DATA;
        }

    PROPERTY
        {
        PR_ROOT *names;      font names list
        PR_ROOT *sizes;      size list for current name
        VOID *wdr;           printer queries
        SCRLAY_FONT *pf;     target for new font data
        SCRLAY_FONT f;       copy of original input font data
        }
    }
```

Property

`fontsel.names` a `VASTR` array containing the names of the typefaces supported by the current printer model

`fontsel.sizes` a `VASTR` array containing font heights in points for the current typeface

`fontsel.wdr`	the handle of an instance of WDR that accesses the current WDR file: a WDR file contains printer specific data including font details. (See the description of the WDR class for further details.)
`fontsel.pf`	a pointer to a SCRLAY_FONT struct which contains the edited font data
`fontsel.f`	a SCRLAY_FONT struct which contains the input font data

Resources

Defined in system resource file *s_.rss*.

```
RESOURCE DIALOG sys_font_dl
    {
    title="Font";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            class=C_CHLIST;
            prompt="Font";
            info=CHLIST{};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Size";
            info=CHLIST{};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Underline";
            info=CHLIST{rid=sys_no_yes;};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Bold";
            info=CHLIST{rid=sys_no_yes;};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Italic";
            info=CHLIST{rid=sys_no_yes;};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Print position";
            info=CHLIST{rid=sys_prn_pos;};
            }
        };
    }

RESOURCE MENU sys_prn_pos
    {
    items=
        {
        CHOICE_ITEM {str="Normal";},
        CHOICE_ITEM {str="Superscript";},
        CHOICE_ITEM {str="Subscript";}
        };
    }
```

Additional items may be appended by sending one or more DL_ITEM_ADD and/or DL_ITEM_APPEND messages during the initialisation of the dialog: see the *Dialog Boxes* chapter for further details.

FONTSEL methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Initialise the items in the dialog. It is assumed that `dlgbox.rbuf` contains the address of a `FONTSEL_DATA` struct.

Writes `dlgbox.rbuf->wdr` to `fontsel.wdr`, writes `dlgbox.rbuf->pf` to `fontsel.pf` and writes `*dlgbox.rbuf->pf` to `fontsel.f`.

Creates an instance of the `VASTR` class, writes its handle to `fontsel.names`, and initialises by sending a `VA_INIT` message to `fontsel.names` specifying a granularity of sixteen.

Creates a second instance of the `VASTR` class, writes its handle to `fontsel.sizes`, and initialises by sending a `VA_INIT` message to `fontsel.sizes` specifying a granularity of sixteen.

Obtains the number of typefaces by sending a `WDR_SENSE_MODEL` message to `fontsel.wdr`. Obtains the name of each typeface by sending a `WDR_TYPEFACE` message to `fontsel.wdr` and adds to the names array by sending `VA_APPEND` messages to `fontsel.names`.

Sets the `fontsel.names` array as the data for the *Font* control by sending `self` a `WN_SET` message.

Sets the `fontsel.sizes` array as the data for the *Size* control by sending `self` a `WN_SET` message.

Obtains the index for the typeface with RTF/Word typeface index `fontsel.pf->typeface` by sending a `WDR_SEARCH_TYPEFACE` message to `fontsel.wdr`, and sets as the selected item in the *Font* control by sending `self` a `WN_SET` message.

Scans the current `WDR` file for the available sizes in the current typeface and adds each size to the sizes array by sending `VA_APPEND` messages to `fontsel.sizes`. (The steps are as described for the `dl_changed` method.)

Sets the selected item in the *Underline* control to *Yes* if the least significant bit in `fontsel.pf->style` is set. Otherwise sets the selected item to *No*.

Sets the selected item in the *Bold* control to *Yes* if the second least significant bit in `fontsel.pf->style` is set. Otherwise sets the selected item to *No*.

Sets the selected item in the *Italic* control to *Yes* if the third least significant bit in `fontsel.pf->style` is set. Otherwise sets the selected item to *No*.

Sets the selected item in the *Print position* control. This may be *Normal*, *Superscript* or *Subscript* depending on whether or not the `WDR_STYLE_SUPER` or `WDR_STYLE_SUB` flags are set in `fontsel.pf->style`: note that these flags are mutually exclusive.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Handle a key press.

Senses the index of the item selected in the *Font* control, converts the index to an RTF/Word index by sending a `WDR_TYPEFACE` message to `fontsel.wdr` and then writes the result to `fontsel.pf->fid`.

Senses the index of the item selected in the *Size* control, obtains the font height in twips by sending a `WDR_FONT_HEIGHT` message to `fontsel.wdr` and then writes the twips height to `fontsel.pf->height`.

Writes zero to `fontsel.pf->style`.

If the selected item in the *Underline* control is *Yes*, sets the least significant bit in `fontsel.pf->style`.

If the selected item in the *Bold* control is *Yes*, sets the second least significant bit in `fontsel.pf->style`.

If the selected item in the *Italic* control is *Yes*, sets the third least significant bit in `fontsel.pf->style`.

Compares `fontsel.pf` and `&fontsel.f` using the `p_bcmp` PLIB library routine writing the return value to `dlgbox.rbuf->ret`: zero indicates identical data.

Returns WN_KEY_CHANGED.

DL_CHANGED — Handle item changed messages

```
VOID dl_changed(INT changed);
```

Set the data and the selected item for the *Size* control.

Obtains the index of the item selected in the *Size* control and then resets the sizes array by sending a VA_RESET message to fontsel.sizes.

Obtains the number of font heights by sending a WDR_TYPEFACE message to fontsel.wdr.

Obtains each font height in twips by sending WDR_FONT_HEIGHT messages to fontsel.wdr, convert into points by dividing by twenty, and adds to the sizes array by sending a VA_APPEND message to fontsel.sizes.

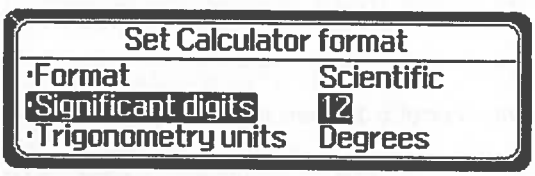
Obtains the index of the font with twips height fontsel.pf->height by sending a WDR_SEARCH_HEIGHT message to fontsel.wdr and then sets this as the index of the selected item in the *Size* control.

EVALDLG

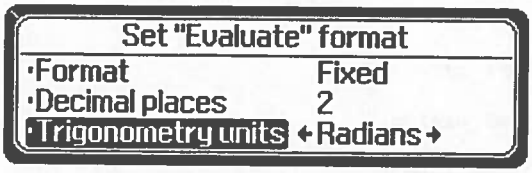
WIN	BWIN	DLGCHAIN	DLGBOX		EVALDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	isEval eMem
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ wn_init	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add ~~dl_set_size~~ dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ ~~dl_changed~~ *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_set_size dl_changed dl_key

The EVALDLG class implements the *Set Calculator format* and *Set "Evaluate" format* dialogs.

The *Set Calculator format* presents the user with the format, number of significant digits and trigonometry units as used by the calculator and which are stored in the system wide M$V environment variable. An example *Set Calculator format* dialog is illustrated in the following picture:



The *Set "Evaluate" format* dialog presents the user with the format, number of decimal places and trigonometry units as used by the evaluator and which are also stored in the system wide M$V environment variable. An example *Set "Evaluate" format* dialog is illustrated in the following picture:



The simplest method of launching either of the above dialogs is to use the `ws_format_dialog` method of the `WSERV` object as described in the *WSERV Class* chapter of the *HWIM Reference* manual.

Class definition

Defined in sub-category file *evaldlg.cl* (generated header file *evaldlg.g*).

```
CLASS evaldlg dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_set_size
    REPLACE dl_changed
    REPLACE dl_key

    CONSTANTS
        {
        EVALDLG_CALCULATOR_EVALUATOR 0 /* set this flag in rbuf for calc evaluator */
        EVALDLG_GENERAL_EVALUATOR  1 /* set this flag in rbuf for general evaluator */
        }
    PROPERTY
        {
        WORD isEval;
        EXTENDED_MEM_VALUES eMem;
        }
    }
```

Property

`evaldlg.isEval`	contains either `EVALDLG_GENERAL_EVALUATOR` for a *Set "Evaluate" format* dialog, or `EVALDLG_CALCULATOR_EVALUATOR` for a *Set Calculator format* dialog
`evaldlg.eMem`	an `EXTENDED_MEM_VALUES` struct that contains the current data

Resources

Defined in system resource file *s_.rss*.

```
RESOURCE DIALOG sys_eval_dialog
    {
    title="Set Calculator format";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            class=C_CHLIST;
            prompt="Format";
            info=CHLIST { rid=sys_format_chlist; };
            },
        CONTROL
            {
            class=C_NCEDIT;
            prompt="Significant digits";
            info=NCEDIT
                {
                high=12;
                low=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Trigonometry units";|
            info=CHLIST { rid=sys_rad_deg; };
            }
        };
    }

RESOURCE MENU sys_format_chlist
    {
    items =
        {
        CHOICE_ITEM { str="Fixed";},
        CHOICE_ITEM { str="Scientific";},
        CHOICE_ITEM { str="General";},
        CHOICE_ITEM { str="Hexadecimal";}
        };
    }

RESOURCE MENU sys_rad_deg
    {
    items =
        {
        CHOICE_ITEM { str= "Radians";},
        CHOICE_ITEM { str= "Degrees";}
        };
    }

RESOURCE STRING sys_evaldlg_title
    {
    str="Set \"Evaluate\" format";
    }
```

EVALDLG methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Initialise the items in the dialog and, if necessary, the M$V environment variable.

Writes `dlgbox.rbuf` to `evaldlg.isEval`: it is assumed that this points to an `EXTENDED_MEM_VALUES` struct which is defined as follows:

```
typedef struct
   {
    UBYTE evalDegrees;
    UBYTE calcDegrees;
    MEM_VALUES memVal;
    } EXTENDED_MEM_VALUES;

typedef struct
    {
    UBYTE evalFormat;  /* Dtob format code for all except calc */
    UBYTE evalDPlaces; /* Decimal places for all except calc */
    UBYTE calcFormat;  /* Dtob format code */
    UBYTE calcDPlaces; /* Decimal places, if relevant */
    DOUBLE values[MAX_MEMORIES];
    } MEM_VALUES;
```

If no `M$V` environment variable exists, creates one with the following default values:

```
p_bfil(p,sizeof(MEM_VALUES),0);
p->evalFormat=(EVAL_DEFAULT_FMT_TYPE|DEGREES_MODE);
p->calcFormat=(CALC_DEFAULT_FMT_TYPE|DEGREES_MODE);
p->evalDPlaces=EVAL_DEFAULT_PLACES;      /* 2  for evaluator */
p->calcDPlaces=CALC_DEFAULT_PLACES;      /* 14 for calc */
```

Writes the contents of the `M$V` environment variable to `evaldlg.eMem.memVal`.

Initialises the remaining members of `evaldlg.eMem` as follows:

```
evaldlg.eMem.evalDegrees=evaldlg.eMem.memVal.evalFormat&DEGREES_MODE;
evaldlg.eMem.calcDegrees=evaldlg.eMem.memVal.calcFormat&DEGREES_MODE;
evaldlg.eMem.memVal.evalFormat&=(~DEGREES_MODE);
evaldlg.eMem.memVal.calcFormat&=(~DEGREES_MODE);
```

For the *Set Calculator format* dialog:

- sets the *Format* control to display the item with index `evaldlg.eMem->memVal.calcFormat`
- sets the *Significant digits* control to display the integer value in `evaldlg.eMem->memVal.evalDPlaces`
- sets the *Trigonometry units* control to display either the item with index one, if `evaldlg.eMem->memVal.calcDegrees` is non-zero, or the item with index zero if it is zero

For the *Set "Evaluate" format* dialog:

- sets the *Format* control to display the item with index `evaldlg.eMem->memVal.evalFormat`
- sets the *Decimal places* control to display the integer value in `evaldlg.eMem->memVal.evalDPlaces`
- sets the *Trigonometry units* control to display the item with index one if `evaldlg.eMem->memVal.evalDegrees` is non-zero. Otherwise sets the control to display the item with index zero.
- sets the dialog title from the text in the `SYS_EVALDLG_TITLE` system resource.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Write the current selection into the `evaldlg.eMem` property and the `M$V` environment variable.

For the *Set "Evaluate" format* dialog:

- writes the index of the item selected in the *Format* control to `evaldlg.isEval.memVal.evalFormat`
- if the format is either *Fixed* or *Scientific*, writes the value in the *Significant digits/Decimal places* control to `evaldlg.isEval.memVal.evalDPlaces`

- writes the index of the item selected in the *Trigonometry units* control to `evaldlg.isEval.memVal.evalDegrees`

For the *Set Calculator format* dialog:

- writes the index of the item selected in the *Format* control to `evaldlg.isEval.memVal.calcFormat`.
- if the format is either *Fixed* or *Scientific*, writes the value in the *Significant digits/Decimal places* control to `evaldlg.isEval.memVal.calcDPlaces`
- writes the index of the item selected in the *Trigonometry units* control to `evaldlg.isEval.memVal.calcDegrees`

Sets the content of `evaldlg.isEval.memval`:

```
pV=&self->evaldlg.isEval;
pV->memVal.calcFormat&=~DEGREES_MODE; /* clear top bit before or'ring */
pV->memVal.calcFormat|=pV->calcDegrees;
pV->memVal.evalFormat&=~DEGREES_MODE;
pV->memVal.evalFormat|=pV->evalDegrees;
```

Writes the content of `evaldlg.isEval.memval` to the `M$V` environment variable: if this does not exist it is created.

Returns `WN_KEY_CHANGED`.

DL_CHANGED — Handle item changed messages

```
VOID dl_changed(INT changed);
```

Set the prompt for the *Significant digits/Decimal places* control and dim or un-dim as appropriate.

If the item selected in the *Format* control is either *General* or *Hexadecimal*, dims the *Significant digits/Decimal places* control.

Otherwise if the item selected in the *Format* control is *Scientific*, sets the the `SYS_SIG_DIGITS` system resource as the prompt for the *Significant digits/Decimal places* control and then un-dims the *Significant digits/Decimal places* control.

Otherwise if the item selected in the *Format* control is *Fixed*, sets the `SYS_DEC_PLACES` system resource as the prompt for the *Significant digits/Decimal places* control and then un-dims the *Significant digits/Decimal places* control.

DL_SET_SIZE — Set minimum default sizes

```
VOID dl_set_size(VOID);
```

Supersend a `DL_SET_SIZE` message.

Sets the prompt for the *Significant digits/Decimal places* control and dims or un-dims the control as appropriate.

Reads the format specified in `evaldlg.eMem.memVal.evalFormat` if the dialog is a *Set "Evaluate" format* dialog, or `evaldlg.eMem.memVal.calcFormat` otherwise.

If the format is either *General* or *Hexadecimal*, dims the *Significant digits/Decimal places* control.

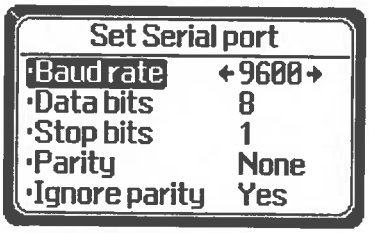
If the format is *Scientific*, sets the the `SYS_SIG_DIGITS` system resource as the prompt for the *Significant digits/Decimal places* control and then un-dims the *Significant digits/Decimal places* control.

If the format is *Fixed*, sets the `SYS_DEC_PLACES` system resource as the prompt for the *Significant digits/Decimal places* control and then un-dims the *Significant digits/Decimal places* control.

SETPORTDLG

WIN	BWIN	DLGCHAIN	DLGBOX		SETPORTDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_key

The SETPORTDLG class implements the *Set serial port* dialog which presents the user with editable serial port characteristics i.e. baud rate, number of data bits, number of stop bits, parity and parity mode. An example *Set serial port* dialog is shown in the following picture:



The example dialog may be obtained by entering the Comms application and selecting the *Port* item in the *Special* menu.

The `dlgbox.rbuf` property *must* point to a `P_SRCHAR` struct which is described in the *Serial port* chapter of the *I/O Devices Reference* manual.

Class definition

Defined in sub-category file *serdlgs.cl* (generated header file *serdlgs.g*).

```
CLASS setportdlg dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

Property

There is no property associated with the SETPORTDLG class.

Resources

Defined in system resource file *s_.rss*.

The Series 3 version of the SYS_BAUD_RATE_CHLIST system resource is as follows:

```
RESOURCE MENU sys_baud_rate_chlist
    {
    items =
        {
        CHOICE_ITEM { str= "9600";},
        CHOICE_ITEM { str= "4800";},
        CHOICE_ITEM { str= "2400";},
        CHOICE_ITEM { str= "1200";},
        CHOICE_ITEM { str= "600";},
        CHOICE_ITEM { str= "300";}
        };
    }
```

The Series 3a version of the SYS_BAUD_RATE_CHLIST system resource is as follows:

```
RESOURCE MENU sys_baud_rate_chlist
    {
    items =
        {
        CHOICE_ITEM { str= "19200";},   /* new for S3a */
        CHOICE_ITEM { str= "9600";},
        CHOICE_ITEM { str= "4800";},
        CHOICE_ITEM { str= "2400";},
        CHOICE_ITEM { str= "1200";},
        CHOICE_ITEM { str= "600";},
        CHOICE_ITEM { str= "300";}
        };
    }
```

The remaining resources are common to both machines:

```
RESOURCE MENU sys_data_bits_chlist
    {
    items =
        {
        CHOICE_ITEM { str="5";},
        CHOICE_ITEM { str="6";},
        CHOICE_ITEM { str="7";},
        CHOICE_ITEM { str="8";}
        };
    }

RESOURCE MENU sys_stop_bits_chlist
    {
    items =
        {
        CHOICE_ITEM { str="1";},
        CHOICE_ITEM { str="2";}
        };
    }

RESOURCE MENU sys_parity_chlist
    {
    items =
        {
        CHOICE_ITEM { str="None";},
        CHOICE_ITEM { str="Even";},
        CHOICE_ITEM { str= "Odd";}
        };
    }
```

```
RESOURCE MENU sys_no_yes
    {
    items =
        {
        CHOICE_ITEM { str= "No";},
        CHOICE_ITEM { str="Yes";}
        };
    }

RESOURCE DIALOG sys_set_port_dialog
    {
    title="Set Serial port";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            prompt="Baud rate";
            info=CHLIST
                {
                rid=sys_baud_rate_chlist;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Data bits";
            info=CHLIST
                {
                rid=sys_data_bits_chlist;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Stop bits";
            info=CHLIST
                {
                rid=sys_stop_bits_chlist;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Parity";
            info=CHLIST
                {
                rid=sys_parity_chlist;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Ignore parity";
            info=CHLIST
                {
                rid=sys_no_yes;
                nsel=0;
                };
            }
        };
    }
```

SETPORTDLG methods

DL_DYN_INIT — Initialise dialog items

```
VOID dl_dyn_init(VOID);
```

Initialise the content of the dialog.

Sets the index of the item selected in the *Baud rate* control according to the flag in `dlgbox.rbuf->tbaud`. The flags and the corresponding indices are as follows:

Series 3 flag	*Series 3a flag*	*index*
P_BAUD_9600	P_BAUD_19200	0
P_BAUD_4800	P_BAUD_9600	1
P_BAUD_2400	P_BAUD_4800	2
P_BAUD_2400	P_BAUD_2400	3
P_BAUD_600	P_BAUD_2400	4
P_BAUD_300	P_BAUD_600	5
none	P_BAUD_300	6

If `dlgbox.rbuf->frame` is non-zero, sets the index of the item selected in the *Data bits* control to `P_DATA_8`.

Otherwise, sets the index of the item selected in the *Data bits* control to zero.

If `dlgbox.rbuf->frame` contains `P_TWO_STOP`, sets the index of the item selected in the *Stop bits* control to one.

If `dlgbox.rbuf->frame` contains the `P_PARITY` flag, sets the index of the item selected in the *Parity* control to `dlgbox.rbuf.parity`.

If `dlgbox.rbuf->frame` contains the `P_IGNORE_PARITY` flag, sets the index of the selected item in the *Ignore parity* control to one i.e. *Yes*.

DL_KEY — Handle key input

```
INT dl_key(WORD id,INT event);
```

Handle a keypress.

Senses the index of the item selected in the *Baud rate* control and then writes the corresponding flag to both `dlgbox.rbuf->tbaud` and `dlgbox.rbuf->rbaud`. The possible indices and the corresponding flags are as follows:

index	*Series 3 flag*	*Series 3a flag*
0	P_BAUD_9600	P_BAUD_19200
1	P_BAUD_4800	P_BAUD_9600
2	P_BAUD_2400	P_BAUD_4800
3	P_BAUD_2400	P_BAUD_2400
4	P_BAUD_600	P_BAUD_2400
5	P_BAUD_300	P_BAUD_600
6	*none*	P_BAUD_300

Writes the index of the item selected in the *Data bits* control to `dlgbox.rbuf->frame`.

If the index of the item selected in the *Stop bits* control is non-zero, ORS the `P_TWOSTOP` flag into `dlgbox.rbuf->frame`.

Writes the index of the item selected in the *Parity* control to `dlgbox.rbuf->parity`, and, if the index is non-zero, ORS `P_PARITY` into `dlgbox.rbuf->frame`.

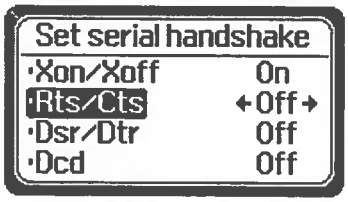
Writes the index of the item selected in the *Ignore parity* control to `dlgbox.rbuf->flags`.

Returns `WN_KEY_CHANGED`.

SETHSHKDLG

WIN	BWIN	DLGCHAIN	DLGBOX		SETHSHKDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_key

The SETHSHKDLG class implements the *Set serial handshake* dialog which present the user with editable serial port handshake settings i.e. the Xon/Xoff, Rts/Cts, Dsr/Dtr and Dcd settings. An example *Set serial handshake* dialog is shown in the following picture:



Each control may be set to either *On* or *Off*.

The `dlgbox.rbuf` property *must* point to a UBYTE containing an ORed combination of handshake flags - the allowed flags are described in the *Serial port* chapter of the *I/O Devices Reference* manual.

Class definition

Defined in sub-category file *serdlgs.cl* (generated header file *serdlgs.g*).

```
CLASS sethshkdlg dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

Property

None.

Resources

Defined in system resource file *s_.rss*.

```
RESOURCE DIALOG sys_set_hshk_dialog
    {
    title="Set serial handshake";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            prompt="Xon/Xoff";
            info=CHLIST
                {
                rid=sys_offon_menu;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Rts/Cts";
            info=CHLIST
                {
                rid=sys_offon_menu;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Dsr/Dtr";
            info=CHLIST
                {
                rid=sys_offon_menu;
                nsel=0;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Dcd";
            info=CHLIST
                {
                rid=sys_offon_menu;
                nsel=0;
                };
            }
        };
    }

RESOURCE MENU sys_offon_menu
    {
    items =
        {
        CHOICE_ITEM { str="Off";},
        CHOICE_ITEM { str= "On";}
        };
    }
```

SETHSHKDLG methods

DL_DYN_INIT — Initialise items

```
VOID dl_dyn_init(VOID);
```

Initialise the items in the dialog.

If `self->dlgbox.rbuf` contains `P_OBEY_XOFF`, sets the *Xon/Xoff* control to display *On*.

If `self->dlgbox.rbuf` does not contain `P_IGN_CTS`, sets the *Rts/Cts* control to display *On*.

If `self->dlgbox.rbuf` contains `P_OBEY_DSR`, sets the *Dsr/Dtr* control to display *On*.

If `self->dlgbox.rbuf` contains `P_OBEY_DCD`, sets the *Dcd* control to display *On*.

DL_KEY — Handle key input

```
INT dl_key(WORD id,INT event);
```

Handle key input.

Write zero to `*dlgbox.rbuf`.

If the index of the item selected in the *Xon/Xoff* control is non-zero, ORS `P_OBEY_XOFF` and `P_SEND_XOFF` into `*dlgbox.rbuf`.

If the index of the item selected in the *Rts/Cts* control is zero, ORS `P_IGN_CTS` into `*dlgbox.rbuf`.

If the index of the item selected in the *Dsr/Dtr* control is non-zero, ORS `P_OBEY_DSR` into `*dlgbox.rbuf`.

If the index of the item selected in the *Dcd* control is non-zero, ORS `P_OBEY_DSD` into `*dlgbox.rbuf`.

Returns `WN_KEY_CHANGED`.

CHAPTER 16

PRINT CLASSES

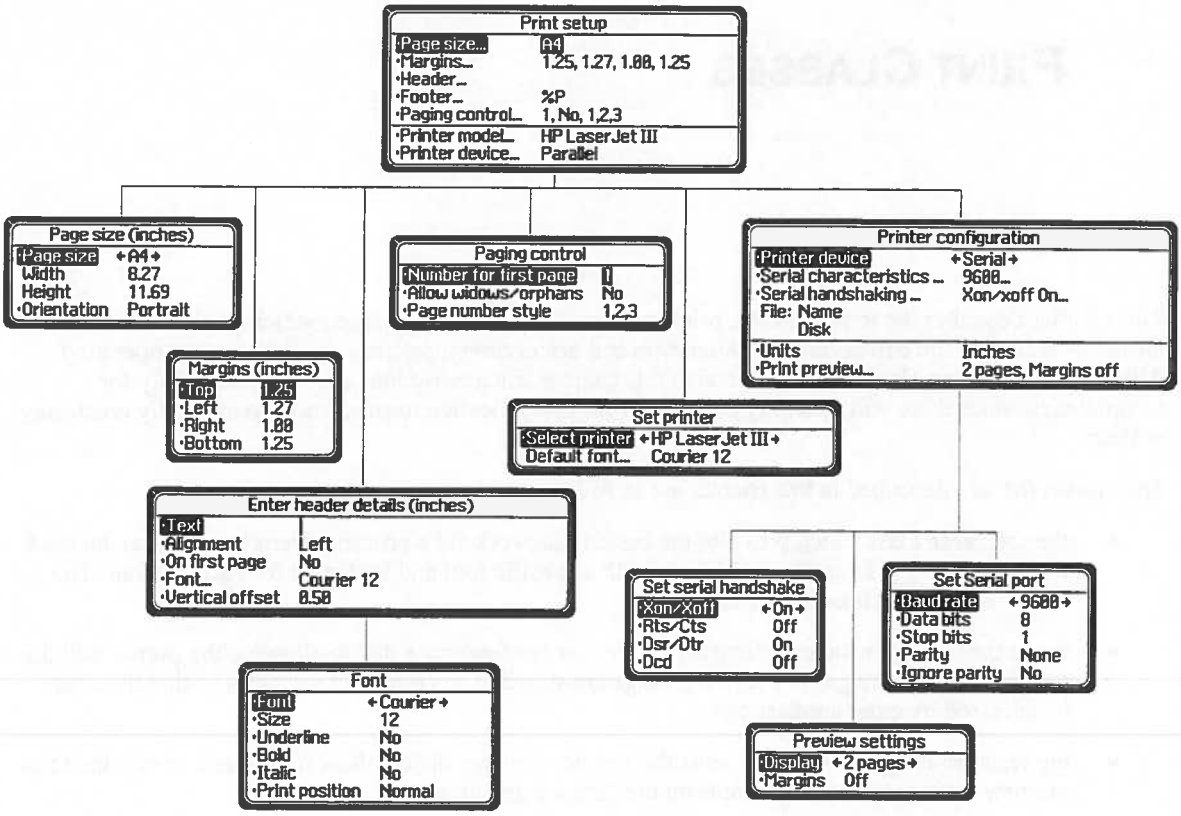
This chapter describes the HWIM printing, print preview and print dialog classes which provide a convenient means of accessing the current printer parameters and performing a printing or print preview operation. Although these dialog classes are described in this chapter, the descriptions are included mainly for completeness since there will generally be no need for an application to subclass or to explicitly create any of them.

The classes that are described in this chapter are as follows:

- the LPRINTER class which provides the basic framework for a printing operation and may be used, for example, to print multiple columns with a specific font and font style for each column. The class must be subclassed to be useful.
- the PDEVDLG class which implements the *Printer configuration* dialog allowing the user to edit the current printer settings. The edited settings are stored as environment variables so that they may be accessed by other applications.
- the PRNPREV class which implements the *Preview settings* dialog allowing the user to edit the print preview settings. It is not available on the Series 3 machine.
- the PRNCTRL class which implements the *Print setup* dialog allowing the user to edit the current page layout settings. The edited settings are stored in the current PRINTER object so that they may be accessed by other objects used by the application.
- the PAGECTRL class which implements the *Paging control* dialog allowing the user to edit the page number style. The edited settings are stored in the current PRINTER object so that they may be accessed by other objects used by the application.
- the MARGINS class which implements the *Margins* dialog allowing the user to edit the page margins. The edited settings are stored in the current PRINTER object so that they may be accessed by other objects used by the application.
- the HEADFOOT class which implements the *Header details* and *Footer details* dialogs allowing the user to edit the header and the footer respectively. The edited settings are stored in the current PRINTER object so that they may be accessed by other objects used by the application.
- the PRNMODEL class which implements the *Set printer* dialog allowing the user to select the printer model and default font. The edited settings are stored in the current PRINTER object so that they may be accessed by other objects used by the application. .
- the PAGESIZE class which implements the *Page size* dialog allowing the user to edit the page size and orientation. The edited settings are stored in the current PRINTER object so that they may be accessed by other objects used by the application.
- the PRINTING class which provides the *Printing* dialog allowing a printing operation to be started, monitored and cancelled. A *Printing* dialog is launched by the LPRINTER class.
- the PMODLOCS class which is used as a component by the PRNMODEL class. This class is highly specialised and is thus unlikely to be used elsewhere.

Any application may create and use the *Print setup* or *Printer configuration* dialogs by sending WS_EDIT_PRINT_CONTEXT or WS_EDIT_PDEV_SETUP messages to the application's instance of (a subclass of) WSERV. All the dialogs described in this chapter are, directly or indirectly, components of either or both of these two dialogs.

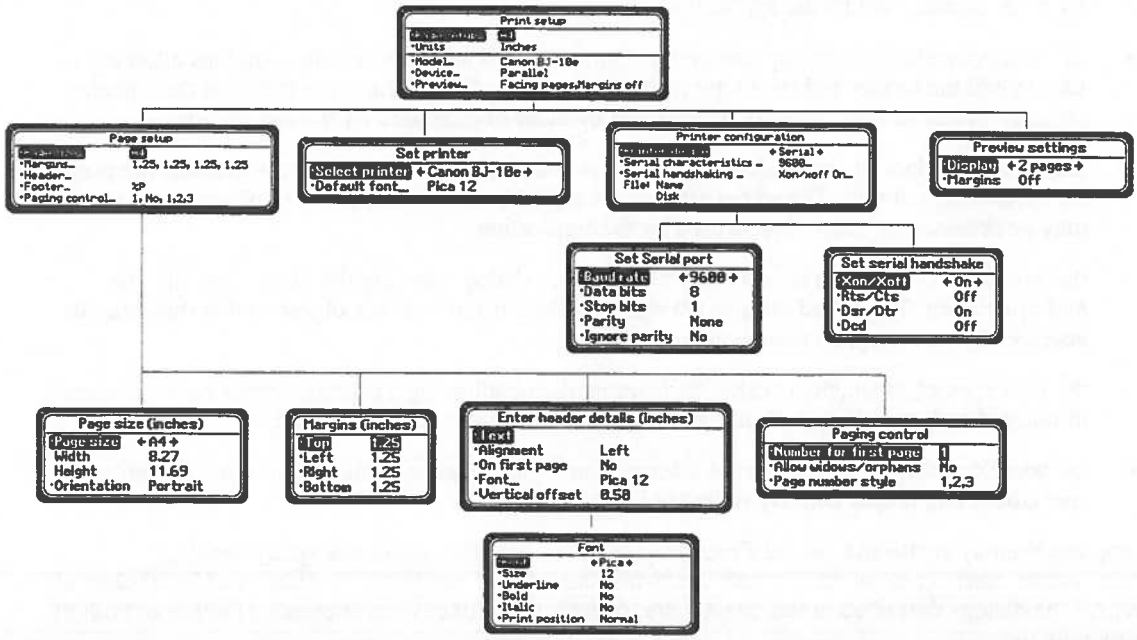
The differences in screen size between the Series 3, Series 3a and Work*about* result in different limits on the number of items that may appear in a dialog. In consequence, there are some differences of implementation of the print dialog classes between the different machine types. On the Series 3a, for example, the dialogs are related as shown below:



All the dialogs are available from the *Print setup* dialog, which would normally be accessed via a *Print setup* option in one of the menus (generally the *Special* menu) of an application that can print data.

Note that some of these dialogs, for example, the *Set serial port* and *Set serial handshake* dialogs, are described in other chapters of this manual.

On the Work*about*, the dialogs are again all accessible from the Print Setup dialog, but the organisation is somewhat different, as illustrated below:



The main differences in appearance are in the *Print setup* and *Page setup* dialogs. The *Print setup* dialog is implemented on the Work*about* by an additional SCPRNCTL class. The *Page setup* dialog contains most of the items that appear in the Series 3a *Print control* dialog. In fact, it is implemented by another new class - SCPGSETUP. This subclasses the PRNCTRL class, replacing only the `dl_dyn_init` method to change the title text and then supersend the DL_DYN_INIT message. These two additional classes are not further documented in this chapter.

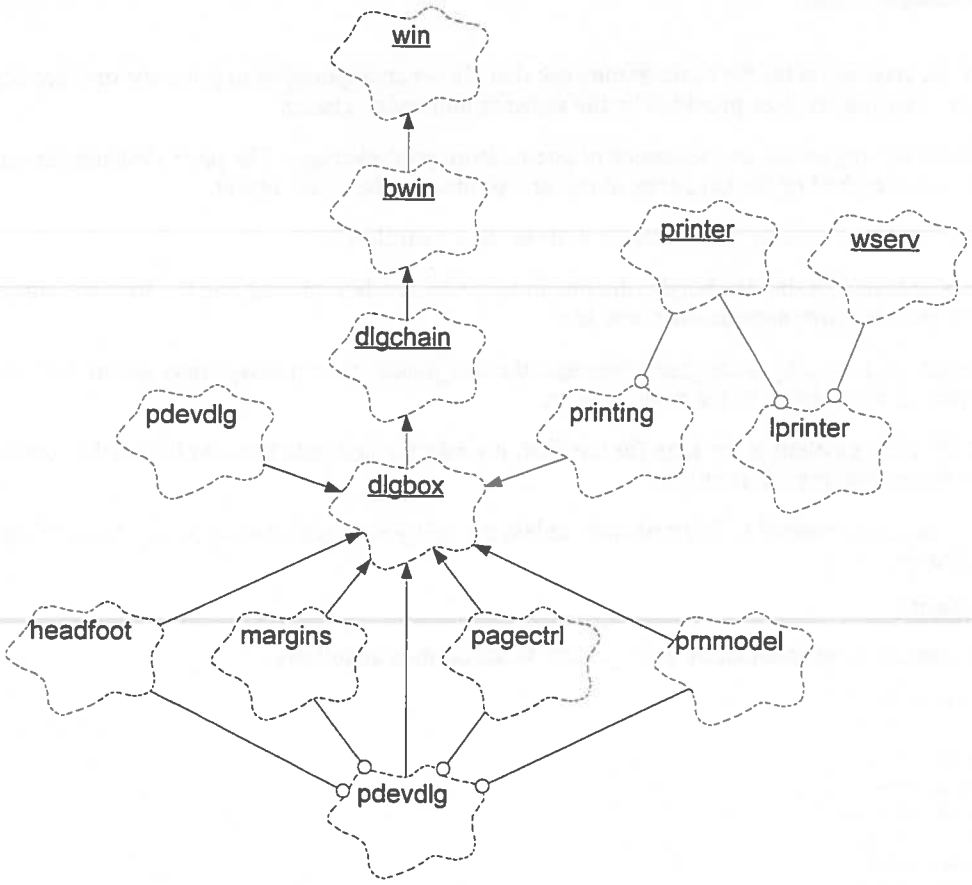
Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the PRINTER class which supplies methods for both editing the current printer settings and starting a printing, pagination or print preview operation.
- the PAGES active object class which is responsible for the printing.
- the LOCS class described in the *OLIB Reference* manual.

The PRINTER and PAGES classes are described in the *Document Printing Classes* of the *FORM Reference* manual.

Class diagram



LPRINTER

LPRINTER
pages wdr f defer wtab lheight width subsqind
destroy lpr_init lpr_read lpr_sense_buf_width *lpr_sense_text*

The LPRINTER class provides the basic framework that allows an application to print one or more copies of a document using the services provided by the PRINTER and PAGES classes.

The document is represented as a sequence of one or more print elements. The print elements are supplied by the lpr_read method of the LPRINTER object and printed by the PAGES object.

The lpr_read method sets the print element in three steps as follows:

- sets defaults for the line height, the line indentation, the line spacing and the font and ensures that the print element appears on a new line.
- sends self an LPR_SENSE_TEXT message: the LPR_SENSE_TEXT message may set the text and any other characteristics of the print element.
- if the print element is too long for one line, it wraps the text onto the next line(s) thus creating two or more shorter print elements.

The lpr_sense_text method is deferred and - unless the lpr_read method is replaced - *must* be supplied by the subclasser.

Print elements

Each print element is represented by a WDR_PRINT struct defined as follows:

```
typedef struct
    {
    WORD flags;
    WORD typf;
    WORD fheight;
    WORD style;
    WORD down;
    WORD indent;
    WORD height;
    WORD right;
    TEXT *buf;
    UWORD blen;
    } WDR_PRINT;
```

The significance of the members of the WDR_PRINT struct is as follows:

flags — an ORed combination of the following flags:

WDR_PRINT_START	this flag indicates that the printer is to be initialised to allow printing.
WDR_PRINT_PAGE	this flag indicates that the print element is on a new page.
WDR_PRINT_LINE	this flag indicates that the print element is on a new line.

WDR_PRINT_FONT	this flag indicates that the current printer font is to be set to the font specified by the typf, height and style members. The current font is set *before* any text is printed.
WDR_PRINT_RIGHT	this flag indicates that the current print position is to be moved right by the number of printer units specified by the right member. The current print position is moved *before* any text is printed.
WDR_PRINT_TEXT	this flag specifies that the text specified by the buf and len members is to be printed.
WDR_PRINT_END	this flag specifies that the print element is the last.

typf — this is the typeface index and uniquely identifies the typeface. This member is ignored unless the flags member contains WDR_PRINT_FONT.

fheight — this is the height of the font in units of twips. This member is ignored unless the flags member contains WDR_PRINT_FONT.

style — this specifies the font style and may contain an ORed combination of the following flags:

WDR_PRINT_NORMAL	plain text.
WDR_PRINT_UNDERLINE	the text is to be printed as underlined text.
WDR_PRINT_BOLD	the text is to be printed as bold text.
WDR_PRINT_ITALIC	the text is to be printed as italic text.
WDR_STYLE_SUPER	the text is to be printed as superscripted text.
WDR_PRINT_SUB	the text is to be printed as subscripted text.

This member is ignored unless the flags member contains WDR_PRINT_FONT.

down — this is the downwards displacement of the print head in printer units. It is automatically *set to zero* by the PAGES object if the print element is at the top of the page. This member is ignored unless the flags member contains WDR_PRINT_LINE.

indent — this is the right indent of the print head in printer units. This member is ignored unless the flags member contains WDR_PRINT_LINE.

height — this is the line height in printer units. When moving to a new line the print head is moved down by the sum of the height and down members. This member is ignored unless the flags member contains WDR_PRINT_LINE.

right — this is the right displacement of the print head in printer units. This member is ignored unless the flags member contains WDR_PRINT_RIGHT.

buf — this is the address of a buffer containing the text to print. This member is ignored unless the flags member contains WDR_PRINT_TEXT.

blen — this is the length of the text to print. This member is ignored unless the flags member contains WDR_PRINT_TEXT.

Class definition

Defined in sub-category file *lprinter.cl* (generated header file *lprinter.g*).

```
CLASS  lprinter root
    {
    REPLACE destroy
    ADD lpr_init
    ADD lpr_read
    ADD lpr_sense_buf_width
    DEFER lpr_sense_text
```

```
PROPERTY
    {
    VOID *pages;            Copy for reference only
    VOID *wdr;              Copy for reference only
    SCRLAY_FONT f;          The default font
    WDR_PRINT defer;        In case previous data was too wide
    UBYTE *wtab;            Width table
    UWORD lheight;          Line height in printer units
    UWORD width;            Width of region to print to
    UWORD subsqind;         Indent for subsequent lines (if wrapped)
    }
}
```

Property

`lprinter.pages` the handle of an instance of the PAGES class which is responsible for the printing. A description of the PAGES class may be found in the *FORM Reference* manual.

`lprinter.wdr` the handle of an instance of the WDR class. This class provides access to the current printer resource file. A description of the WDR class may be found in the *FORM Reference* manual.

`lprinter.f` a pointer to a SCRLAY_FONT struct which contains details of the default font. The SCRLAY_FONT struct is defined as follows:

```
typedef struct
    {
    UWORD fid;
    UWORD style;
    UWORD height;
    } SCRLAY_FONT;
```

The significance of the members of the above struct is as follows:

`fid` this is the default font ID.

`style` this is the default font style.

`height` this is the default font height in units of twips.

The default font characteristics are obtained from the current PRINTER object.

`lprinter.defer` a WDR_PRINT struct containing details of deferred text. Deferred text results from the `lpr_sense_text` method providing too much text for one line. The deferred text is printed on subsequent line(s) with a line indentation of `lprinter.subsqind`.

`lprinter.wtab` a pointer to the current printer width table.

A monospace font table contains two bytes. The first byte is always equal to zero. The second byte specifies the width of a character .

A proportional font table contains one byte per character. The width of the character with code `c` is specified by the byte with offset `c`.

`lprinter.lheight` specifies the default line height in printer units used by the `lpr_read` method.

`lprinter.width` specifies the width of one line in printer units used by the `lpr_read` method.

`lprinter.subsqind` specifies the line indentation to be used when printing deferred text.

LPRINTER methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the LPRINTER instance.

If w_ws->wserv.printer is non-zero, sends a PR_CLOSE_WDR message to w_ws->wserv.printer.

Supersends a DESTROY message.

LPR_INIT — Start printing dialog

```
VOID lpr_init(VOID);
```

Initialise the LPRINTER object property and then start the printing operation.

Ensures that an instance of the PRINTER printer manager class exists by sending a WS_ENS_PRINT_CONTEXT message to w_ws.

Obtains a pointer to a PRINTER_PARAMS struct - which contains the current printer parameters - by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Stores the width of the printing region in property by writing the p.pg.body.width member of the PRINTER_PARAMS struct to lprinter.width.

Stores the default printer font in property by writing the d.f member of the PRINTER_PARAMS struct to lprinter.f.

Not on the Series 3: if lprinter.subsqind is non-zero, writes FALSE to lprinter.subsqind and returns. A non-zero value for lprinter.subsqind indicates that the LPRINTER object is being used for a print preview operation.

Specifies the PAGES active object that is to handle the printing by writing lprinter.pages to the ppages member of an RBUF_PRINTING struct.

Specifies the call-back handle for reading the next line of text - in the form of a WDR_PRINT struct - by writing self to the calls hread member of the above RBUF_PRINTING struct.

Specifies the call-back message for reading the next line of text by writing O_LPR_READ to the calls hread member of the above RBUF_PRINTING struct.

Starts the printing operation by launching a *Printing* dialog with the address of the above RBUF_PRINTING struct as the data - details of the *Printing* dialog and the RBUF_PRINTING struct may be found in the description of the PRINTING class.

LPR_READ — Read call-back

```
VOID lpr_read(INT x,WDR_PRINT *pr);
```

Write to the WDR_PRINT struct specified by pr details of the next portion of text to print.

This call-back method is called by the PAGES active object when it requires the next print element.

The first step is to ensure that all property has been initialised. Thus if lprinter.wdr is zero:

- sets the default line height as specified by lprinter.lheight to be equal to the height of the default font specified by lprinter.f.height.
- converts the default line height - lprinter.lheight - into vertical printer units.
- converts the default line width - lprinter.width - into horizontal printer units.
- sets the handle of the WDR printer resource object specified by lprinter.wdr to the pages.in.wdr member of lprinter.pages - this is assumed be the handle of a WDR object used by the current PAGES object.
- writes the address of the printer width table for the default font specified by lprinter.f to lprinter.wtab - the method obtains the address by sending a WDR_GET_WIDTH_TABLE message to lprinter.wdr.

If lprinter.defer.blen is non-zero, the content of the WDR_PRINT struct pointed to by pr is set from the deferred text specified by lprinter.defer:

- writes lprinter.defer to *pr.
- sets the line indentation specified by pr->indent to lprinter.subsqind.

Otherwise the content of the WDR_PRINT struct pointed to by pr is set as follows:

- writtes default values for the font, line height, line indentation and line separation and indicates that the print element is to appear on a separate line:

```
pr->flags=WDR_PRINT_FONT|WDR_PRINT_LINE|WDR_PRINT_TEXT;
pr->typf=self->lprinter.f.fid;
pr->fheight=self->lprinter.f.height;
pr->style=self->lprinter.f.style;
pr->height=self->lprinter.lheight;
pr->down      =0;
pr->indent=0;
```

- sends self an LPR_SENSE_TEXT message with an argument of pr.
- if the LPR_SENSE_TEXT message indicated that the printing is complete - i.e. the return value is FALSE - writes WDR_PRINT_END to pr->flags and then returns.

Writes *pr to lprinter.defer.

If the line indentation specified by pr->indent exceeds the line width specified by lprinter.width, the indentation is reset to zero.

Determines the number of characters in the buffer pointed to by pr->buf that will fit on the current line assuming the current font and line width. The current line is broken at a word boundary if at all possible.

The difference between the result and the value specified by pr->len defines the length of any deferred text i.e. text that wraps onto one or more subsequent lines.

Writes the length of the text that will fit on the current line to pr->blen.

Writes the length of the deferred text to lprinter.defer.blen.

If there is deferred text:

- writes the address of the first deferred character to lprinter.defer.buf.
- writes the length of the deferred text to lprinter.defer.blen.
- *not on the Series 3*: sets WDR_PRINT_LINE in lprinter.defer.flags.
- obtains the address of a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.
- if the d.wo_control member of the PRINTER_PARAMS struct is zero - indicating that widows and orphans are not allowed - sets WDR_PRINT_KEEP in pr->flags.

LPR_SENSE_BUF_WIDTH — Get printer text width

```
INT lpr_sense_buf_width(TEXT *buf,INT len);
```

Return the width in printer units of the first len characters pointed to by buf.

The method evaluates the length of the text using the current printer width table at address lprinter.wtab.

Deferred LPRINTER methods

LPR_SENSE_TEXT — Get text to print

```
INT lpr_sense_text(WDR_PRINT *pr);
```

Get the next portion of text to print - this method is called by the lpr_read method

For the simplest printing requirements the replaced method should write the address and length of the text to print to pr->buf and pr->len respectively.

For more sophisticated printing requirements the default line height, line indentation, line spacing and font may be overridden by writing appropriate values to the appropriate members of the WDR_PRINT struct pointed to by pr.

In all cases the replaced method should return FALSE if no more text remains.

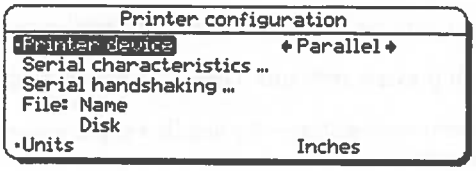
Note that it may be better to replace the lpr_read method, and ignore the lpr_sense_text method, if a significant amount of processing would otherwise have to be carried out in the lpr_sense_text method.

PDEVDLG

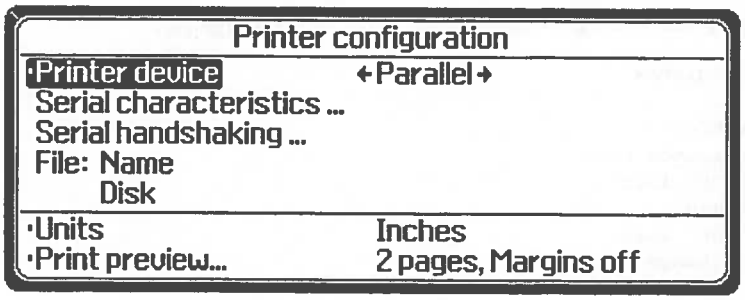
WIN	BWIN	DLGCHAIN	DLGBOX		PDEVDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	printer srchar filechanged own_printer Disp
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~*wn_set*~~ ~~*wn_sense*~~ ~~*wn_draw*~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add ~~dl_set_size~~ dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ ~~*dl_changed*~~ *dl_focus* ~~*dl_launch_sub*~~ *dl_item_new*	dl_launch_sub dl_dyn_init dl_key dl_set_size dl_changed

The PDEVDLG class implements the *Printer configuration* dialog allowing the user to select the printer device and edit the associated settings.

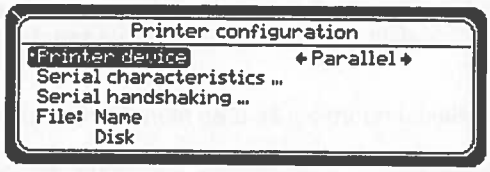
An example *Printer configuration* dialog for the Series 3 is shown in the following picture:



and examples of the equivalent dialogs for the Series 3a and Work*about* are shown below:



Series 3a *Printer configuration* dialog



Work*about Printer configuration* dialog

For the Series 3a dialog, the important difference is the extra *Print preview* control which allows the user to edit the print preview settings. On the Work*about*, this additional facility is provided from the *Print setup* dialog, as indicated at the beginning of this chapter.

The edited values - with the exception of the printer units - are stored in system-wide environment variables as follows:

`P$D` this stores the printer port type and may be set to one of the following values:

- `PRINTER_PORT_PARALLEL` indicates that the current printer device is the parallel port.
- `PRINTER_PORT_SERIAL` indicates that the current printer device is the serial port.
- `PRINTER_PORT_FILE` indicates that the current printer device is a file.
- `PRINTER_PORT_FAX` indicates that the current printer device is the fax.

`P$F` this stores the full file specification of a file to which printing may be directed. The name is stored as a zero terminated string

`P$S` this stores the serial port characteristics organised as a `P_SRCHAR` struct.

`P$P` Not used the Series 3. This environment variable, which is four bytes long, stores the print preview settings as a zero terminated string containing two characters.

The first character specifies the print preview mode as follows:

- '0' specifies that print preview operations are to display facing pages.
- '1' specifies that print preview operations are to display one page.
- '2" specifies that print preview operations are to display two pages.
- '3' specifies that print preview operations are to display three pages.
- '4' specifies that print preview operations are to display four pages.

The second byte specifes the margins mode as follows:

- '0' specifies that print preview operations *are not* to display the margins.
- '1' specifies that print preview operations *are* to display the margins.

The printer units is stored as a system wide setting - for details see the `p_getctd` and `p_setctd` PLIB routines in the *PLIB Reference* manual.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS  pdevdlg  dlgbox
    {
    REPLACE destroy
    REPLACE dl_launch_sub
    REPLACE dl_dyn_init
    REPLACE dl_key
    REPLACE dl_set_size
    REPLACE dl_changed
```

```
    PROPERTY 1
        {
        PR_ROOT *printer;
        P_SRCHAR srchar;
        WORD filechanged;
        WORD own_printer;
        PVV_DISPLAY Disp;
        }
    }
```

Property

`pdevdlg.printer`	The handle of an instance of the PRINTER class. This object supports the setting and sensing of data required for print, pagination and print preview operations.
`pdevdlg.srchar`	A P_SRCHAR struct containing the current serial port characteristics. These are either copied from the P$F environment variable, or, if this does not exist, set to default values.
`pdevdlg.filechanged`	Set to TRUE if the user has edited either the characteristics associated with one or more devices or the printer units - selecting a new printer device does *not* set this property to TRUE. The default value is FALSE.
`pdevdlg.own_printer`	Set to TRUE if the PDEVDLG object creates its own PRINTER object on initialisation. Set to FALSE if the PDEVDLG object uses the PRINTER object with handle `w_ws->wserv.printer`. This property is not available on the Series 3.
`pdevdlg.Disp`	Contains the current print preview settings stored as a PVV_DISPLAY struct - see the description of the PRNPREV class for details. This property is not available on the Series 3.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE CONTROL sys_print_preview_settings
    {
    class=C_TEXTWIN;
    prompt="Print preview"<WS_SYMBOL_ELLIPSIS>;
    info=TXTMESS
        {
        str="Facing pages,Margins off";        /* widest possible text */
        flags=IN_TEXTWIN_POPOUT;
        };
    }

RESOURCE DIALOG sys_printer_config_dialog
    {
    title="Printer configuration";
    flags=DLGBOX_NOTIFY_ENTER;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            prompt="Printer device";
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            info=CHLIST
                {
                rid=sys_printer_device_chlist;
                };
            },
```

```
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Serial characteristics "<WS_SYMBOL_ELLIPSIS>;
            info=TXTMESS
                {
                str="19200 "<WS_SYMBOL_ELLIPSIS>;       /* changed for S3a */
                flags=IN_TEXTWIN_POPOUT;
                };
            },

        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Serial handshaking "<WS_SYMBOL_ELLIPSIS>;
            info=TXTMESS
                {
                str="Xon/Xoff Off "<WS_SYMBOL_ELLIPSIS>;
                flags=IN_TEXTWIN_POPOUT;
                };
            },

        CONTROL
            {
            flags=DLGBOX_ITEM_NEEDS_PACK|DLGBOX_ITEM_NOTIFY_CHANGED;
            class=C_FNEDIT;
            prompt="File:";
            info=FNEDIT { flags=IN_FNEDIT_NO_AUTOQUERY; };
            },

        CONTROL
            {
            class=C_CHLIST;
            prompt="Units";
            info=CHLIST
                {
                rid=sys_printer_units_chlist;
                };
            }
        };
    }
```

Note: on the Series 3, the text "19200" in the *Serial characteristics* control is replaced by "9600".

PDEVDLG methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the PDEVDLG object.

If `pdevdlg.own_printer` is non-zero, sends a destroy messsage to `pdevdlg.printer`.

Supersends a DESTROY message.

This method is not replaced on the Series 3.

DL_LAUNCH_SUB — Launch sub-dialog

```
VOID dl_launch_sub(INT index);
```

Launch either a *Set serial port* dialog or a *Set serial handshake* dialog according to the value of `index`.

If `index` is two, launches a *Set serial port* dialog - details of this dialog may be found in the *General dialogs* chapter of the *HWIM Reference* manual - in order to edit some of the serial port characteristics stored in `pdevdlg.srchar`.

Otherwise, launches a *Set serial handshake* dialog - details of this dialog may be found in the *General dialogs* chapter of the *HWIM Reference* manual - in order to edit the serial port handshake settings stored in `pdevdlg.srchar.hand`.

Stores the new serial port settings in the `P$S` environment variable by sending a `PR_STORE_CHAR` message to `pdevdlg.printer` specifying the address of `pdevdlg.srchar` as argument.

Sets the *Serial characteristics* control to display appropriate text: the text is either the `NULL` string, or the appropriate item in the `SYS_BAUD_RATE_CHLIST` system resource followed by an ellipsis symbol.

Sets the *Serial handshake* control to display appropriate text: the text is either the `NULL` string, or the concatenation of the `SYS_XONXOFF_STRING`, an appropriate item from the `SYS_OFFON_MENU` system resource and an ellipsis symbol.

The following actions are not performed on the Series 3.

If `index` is seven, allows the user to edit the print preview settings stored in `pdevdlg.Disp` by launching a *Preview settings* dialog.

Stores the edited print preview settings as a zero terminated string containing two characters in the `P$P` environment variable. See the introduction for details of the `P$P` environment variable.

Sets appropriate text in the *Print preview* control.

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog.

The following actions are not performed on the Series 3:

- enables appropriate context sensitive help by writing `-SYS_HELP_PRINT` to `dlgbox.helprid`.
- if `w_ws->wserv.printer` is non-zero - and is thus assumed to store the handle of the current printer object - writes `w_ws->wserv.printer` to `pdevdlg.printer`.
- otherwise writes `TRUE` to `pdevdlg.own_printer`, creates an instance of the `PRINTER` class, writes its handle to `pdevdlg.printer` and then initialises the `PRINTER` object by sending a `PR_INIT` message to `pdevdlg.printer`.

The following actions are only performed on the Series 3:

- creates an instance of the `PRINTER` class, writes its handle to `pdevdlg.printer` and then initialises the `PRINTER` object by sending a `PR_INIT` message to `pdevdlg.printer`.

Obtains the current serial port characteristics - in the form of a `P_SRCHAR` struct - and the name of the current file to which printing is directed by sending a `PR_PORT_DATA` message to `pdevdlg.printer`.

Writes the current serial port characteristics to `pdevdlg.srchar`.

Sets the *File name* control to display the name of the file to which printing is directed.

Obtains the current system wide units - i.e. either `E_METRIC` or `E_IMPERIAL` - and sets this as the index of the item selected in the *Units* control - for further details see the description of the `E_CONFIG` struct in the *PLIB Reference* manual.

If the item with index one is selected in the *Printer device* control - i.e. the *Serial* item - undims the *Serial characteristics* and *Serial handshaking* controls and then dims the *File name* control.

If the item with index two is selected in the *Printer device* control - i.e. the *File* item - dims the *Serial characteristics* and *Serial handshaking* controls and then undims the *File name* control.

Otherwise, dims the *Serial characteristics*, *Serial handshaking* and *File name* controls.

The following actions areonly performed on the Series 3.

If `w_ws->wserv.flags` contains `PR_WSERV_FULLSCREEN`:

- adds to the dialog an underline separating the *Disk* and *Units* controls, and then appends to the dialog the *Print preview* control defined by the `SYS_PRINT_PREVIEW_SETTINGS` system resource.

- retrieves the print preview settings stored in the P$P environment variable and writes appropriate values to dpevdlg.Disp.

DL_KEY Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the current selections as system wide settings.

Senses the index of the item selected in the *Printer device* control and sets the result into the P$D environment variable by sending a PR_SET_PORT_TYPE message to pdevdlg.printer.

If pdevdlg.filechanged is TRUE, senses the file name displayed in the *File name* control and sets the result into the P$F environment variable by sending a PR_STORE_FILE message to pdevdlg.printer.

Senses the index of the item selected in the *Units* control and sets this as the system wide units - for further details see the description of the E_CONFIG struct in the *PLIB Reference* manual.

Not on the Series 3: if the system units are currently set to metric, sets PR_WSERV_METRIC in w_ws->wserv.flags, otherwise clears PR_WSERV_METRIC in w_ws->wserv.flags.

Returns WN_KEY_CHANGED.

DL_SET_SIZE Read in string resources

```
VOID dl_set_size(VOID);
```

Set the size of the dialog and the text in the *Serial characteristics* and *Serial handshaking* controls.

Supersends a DL_SET_SIZE message.

Sets appropriate text in the *Serial characteristics* control - the text is created by concatenating an appropriate item from the SYS_BAUD_RATE_CHLIST system resource and an ellipsis symbol.

Sets appropriate text in the *Serial handshaking* control - the text is created by concatenating the SYS_XONXOFF_STRING system string resource, an appropriate item from the SYS_OFFON_MENU system resource and an ellipsis character.

Not on the Series 3: sets appropriate text in the *Print preview* control.

DL_CHANGED Handle item changed messages

```
VOID dl_changed(INT index);
```

Set the dim status of the *Serial characteristics*, *Serial handshaking* and *File name* controls according to the item selected in the *Printer device* control.

If index is one - i.e. the content of the *Printer device* control has changed - senses the index of the item selected in the Printer device control and

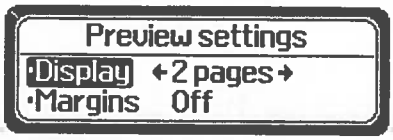
- if the index is one, i.e. the *Serial* item, undims the *Serial characteristics* and *Serial handshaking* controls and then dims the *File name* control.
- if the index is two, i.e. the *File* item, dims the *Serial characteristics* and *Serial handshaking* control and undims the *File name* control.
- otherwise, dims the *Serial characteristics*, *Serial handshaking* and *File name* controls.

Otherwise, indicates that an item in the dialog has changed by writing TRUE to pdevdlg.filechanged.

PRNPREV

WIN	BWIN	DLGCHAIN	DLGBOX		PRNPREV
flags id		next	item rbuf dimrid helprid flags focus count current underline absorb changed	item rbuf dimrid helprid flags focus count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_key

The PRNPREV class implements the *Preview settings* dialog allowing the user to edit the current settings for print preview operations. An example *Preview settings* dialog is shown in the following picture:



The PRNPREV class is not available on Series 3 machines.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS prnprev dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

Property

There is no property associated with the PRNPREV class.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE MENU sys_offon_menu
    {
    items =
        {
        CHOICE_ITEM { str="Off";},
        CHOICE_ITEM { str= "On";}
        };
    }
```

Defined in the system resource file *sx_.ra*.

```
RESOURCE MENU sys_preview_options_chlist
    {
    items=
        {
        CHOICE_ITEM {str="Facing pages";},
        CHOICE_ITEM {str="1 page";},
        CHOICE_ITEM {str="2 pages";},
        CHOICE_ITEM {str="3 pages";},
        CHOICE_ITEM {str="4 pages";}
        };
    }

RESOURCE DIALOG sys_preview_settings_dl
    {
    title="Preview settings";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            prompt="Display";
            info=CHLIST{rid=sys_preview_options_chlist;};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Margins";
            info=CHLIST{rid=sys_offon_menu;};
            }
        };
    }
```

PRNPREV methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog from the `PVV_DISPLAY` struct pointed to by `dlgbox.rbuf`.

The `PVV_DISPLAY` struct is defined as follows:

```
typedef struct
    {
    UBYTE Mode;
    UBYTE Flags;
    } PVV_DISPLAY;
```

The significance of the members of the struct is as follows:

`Mode` this may take a value from zero to four inclusive to specify that either facing pages, one page, two pages, three pages or four pages are to be displayed in print preview operations.

`Flags` this may be `TRUE` to specify that the margins are to be shown in print preview operations. It should be `FALSE` otherwise.

Set the index of the item selected in the *Display* control to `dlgbox.rbuf->Mode`.

Set the index of the item selected in the *Margins* control to `dlgbox.rbuf->Flags`.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the edited content of the dialog in the PVV_DISPLAY struct pointed to by `dlgbox.rbuf`.

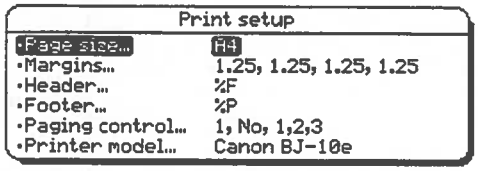
Sense the index of the item selected in the *Display* control and write the result to `dlgbox.rbuf->Mode`.

Sense the index of the item selected in the *Margins* control and write the result to `dlgbox.rbuf->Flags`.

PRNCTRL

WIN	BWIN	DLGCHAIN	DLGBOX		PRNCTRL
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	tm
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size ~~dl_inq_minsize~~ ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* ~~dl_launch_sub~~ *dl_item_new*	dl_dyn_init dl_launch_sub dl_inq_minsize dl_key

The PRNCTRL class implements the *Print setup* dialog allowing the user to edit the current page layout settings. An example *Print setup* dialog is shown in the following picture:



The user may press the Tab key after having selected:

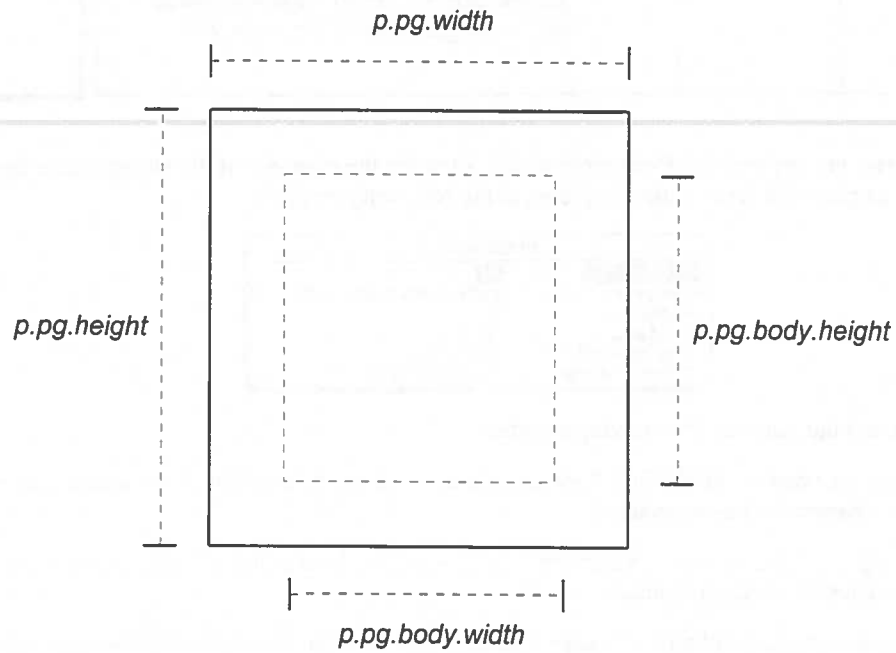
- the *Page size* control to obtain a *Page size* dialog - see the description of the PAGESIZE class in the current chapter for further details.
- the *Margins* control to obtain a *Margins* dialog - see the description of the MARGINS class in the current chapter for further details.
- the *Header* control to obtain a *Header details* dialog - see the description of the HEADFOOT class in the current chapter for further details.
- the *Footer* control to obtain a Footer details dialog - see the description of the HEADFOOT class in the current chapter for further details.

- the *Paging* control to obtain a *Paging control* dialog - see the description of the PAGECTRL class in the current chapter for further details.
- the *Printer model* control to obtain a *Set printer* dialog - see the description of the PRNMODEL class in the current chapter for further details.

The members of the PRINTER_PARAMS struct relevant to the PRNCTRL class are as follows:

`p.pdrflags`	may contain WDR_PDR_LANDSCAPE to specify landscape page orientation. By default the page orientation is portrait.
`p.pg.width`	specifes the total page width in units of twips.
`p.pg.height`	specifies the total page height in units of twips.
`p.pg.body.tl.x`	specifies the horizontal offset of the printing region from the left edge of the page in units of twips.
`p.pg.body.tl.y`	specifies the vertical offset of the printing region from the top edge of the page in units of twips.
`p.pg.body.width`	specifies the width of the printing region - i.e. the page width minus the width of the left and right margins - in units of twips.
`p.pg.body.height`	specifies the height of the printing region - i.e. the page height minus the height of the top and bottom margins - in units of twips.
`p.pgnum.style`	specifies the index of an item in the SYS_PGNO_CHOICE system resource and the style used for page numbers.
`p.pgnum.offset`	specifies the page number offset i.e. the value by which all page numbers are to be offset.
`d.size_choice`	specifies the the index of an item in the SYS_PAGE_SIZE system resource containing the name of the current page size e.g. *A4*.
`d.wo_control`	specifies whether widows and orphans are allowed and may be either TRUE or FALSE.

Some of the above members are illustrated more clearly in the following picture:



The outer square represents the page whilst the inner one represents the printing region i.e. the page minus the margins.

The page layout settings are stored in the curent PRINTER object the handle of which *must* be stored in `w_ws->wserv.printer`.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS prnctrl dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_launch_sub
    REPLACE dl_inq_minsize
    REPLACE dl_key

    TYPES
        {
        typedef struct
            {
            UWORD    tmarg;
            UWORD    lmarg;
            UWORD    rmarg;
            UWORD    bmarg;
            } TMARG;
        }

    PROPERTY
        {
        TMARG   tm;
        }
    }
```

Property

`prnctrl.tm` a `TMARG` struct used to temporarily store the page margins. The `TMARG` struct is defined as follows:

```
typedef struct
{
unsigned short int  tmarg;
unsigned short int  lmarg;
unsigned short int  rmarg;
unsigned short int  bmarg;
} TMARG;
```

`tmarg` specifies the height in twips of the top margin.

`lmarg` specifies the width in twips of the left margin.

`rmarg` member specifies the width in twips of the right margin.

`bmarg` specifies the height in twips of the bottom margin.

Resources

Defined in the system resource file *sx_.ra*.

```
RESOURCE CONTROL sys_print_control_device
    {
    class=C_TEXTWIN;
    prompt="Printer device"<WS_SYMBOL_ELLIPSIS>;
    info=TXTMESS
        {
        str="";
        flags=IN_TEXTWIN_POPOUT;
        };
    }
```

Defined in the system resource file *s_.rss*.

```
RESOURCE DIALOG sys_print_control_dl
    {
    title="Print setup";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_NOTIFY_ESCAPE;
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Page size ";
            info=TXTMESS
                {
                str="";
                flags=IN_TEXTWIN_POPOUT;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Margins ";
            info=TXTMESS
                {
                str="";
                flags=IN_TEXTWIN_POPOUT;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Header ";
            info=TXTMESS
                {
                str="";
                flags=IN_TEXTWIN_POPOUT;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Footer ";
            info=TXTMESS
                {
                str="";
                flags=IN_TEXTWIN_POPOUT;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Paging control ";
            info=TXTMESS
                {
                str="";
                flags=IN_TEXTWIN_POPOUT;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Printer model ";
            info=TXTMESS
                {
                str="";
                flags=IN_TEXTWIN_POPOUT;
                };
            }
        };
    }
```

PRNCTRL methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog.

Obtains a pointer to a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Writes the page margins to prnctrl.tm.

Sets the text in the *Page size* control according to the value stored in the d.size_choice member of the PRINTER_PARAMS struct.

Sets the text in the *Margins* control according to the content of prnctrl.tm.

Sets the header text in the *Header* control: the header text is obtained by sending a PR_GET_HD message to w_ws->wserv.printer.

Sets the footer text in the *Footer* control: the footer text is obtained by sending a PR_GET_HD message to w_ws->wserv.printer.

Sets the text in the *Paging control* control according to the values stored in the p.pgnum.offset, d.wo_control and p.pgnum.style members of the PRINTER_PARAMS struct.

Opens the current WDR file by sending PR_SENSE_MODEL and PR_OPEN_WDR messages to the PRINTER object and then senses the name of the printer model by sending a WDR_SENSE_MODEL_NAME message to the WDR object.

Sets the name of the printer model as the text in the *Printer model* control and then closes the current WDR file by sending a PR_CLOSE_WDR message to the PRINTER object.

The following actions are not performed on the Series 3.

If w_ws->wserv.flags contains PR_WSERV_FULLSCREEN:

- adds a horizontal line between the *Paging control* and *Printer model* controls.
- appends a *Printer device* control as defined by the SYS_PRINT_CONTROL_DEVICE system resource.
- if the *Printer device* displays neither *Parallel*, nor *Serial*, nor *File*, locks the *Printer device* control using the hDlgItemLock utility routine.

Enables content sensitive help by writing -SYS_HELP_PRINT to dlgbox.helprid.

DL_LAUNCH_SUB — Launch sub-dialog

```
VOID dl_launch_sub(INT index);
```

Launch an appropriate sub-dialog.

If index is one, launches a *Page size* dialog - for details see the description of the PAGESIZE class in the current chapter.

If index is two, launches a *Margins* dialog - for details see the description of the MARGINS class in the current chapter. The dialog edits the content of prnctrl.tm.

If index is three, launches a *Header* dialog - for details see the description of the HEADFOOT class in the current chapter.

If index is four, launches a *Footer* dialog - for details see the description of the HEADFOOT class in the current chapter.

If index is five, launches a *Paging control* dialog - for details see the description of the PAGECTRL class in the current chapter.

If index is six, launches a *Printer model* dialog - for details see the description of the PRNMODEL class in the current chapter.

Sets the text in the *Page size* control according to the value stored in the d.size_choice member of the PRINTER_PARAMS struct.

Sets the text in the *Margins* control according to the content of prnctrl.tm.

Sets the text in the *Header* control: the header text is obtained by sending a PR_GET_HD message to w_ws->wserv.printer.

Sets the text in the *Footer* control: the footer text is obtained by sending a PR_GET_HD message to w_ws->wserv.printer.

Sets the text in the *Paging control* control according to the values stored in the p.pgnum.offset, d.wo_control and p.pgnum.style members of the PRINTER_PARAMS struct.

Opens the current WDR file by sending PR_SENSE_MODEL and PR_OPEN_WDR messages to the PRINTER object and then senses the name of the printer model by sending a WDR_SENSE_MODEL_NAME message to the WDR object.

Sets the name of the printer model as the text in the *Printer model* control and then closes the current WDR file by sending a PR_CLOSE_WDR message to the PRINTER object.

DL_INQ_MINSIZE — Change default minimum sizes

```
VOID dl_inq_minsize(INT *pOverallWidth,INT *pPromptWidth,INT *pControlWidth);
```

Write the width of the control to the INT pointed to by pControlWidth.

If w_ws->wserv.flags contains PR_WSERV_FULLSCREEN, writes 160 to address pControlWidth.

Otherwise writes 120 to address pControlWidth.

The pOverallWidth and pPromptWidth arguments are not used.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the edited content of the dialog in the PRINTER object.

Obtains a pointer to a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Writes the offset of the printing region to the p.pg.body.tl member of the PRINTER_PARAMS struct.

Writes the width of the printing region to the p.pg.body.width member of the PRINTER_PARAMS struct.

Writes the height of the printing region to the p.pg.body.height member of the PRINTER_PARAMS struct.

Not on the Series 3: if the margins are too wide/high for the page:

- sets the width of the printing region equal to the width of the page: writes the p.pg.width member of the PRINTER_PARAMS struct to the p.pg.body.width member.
- sets the height of the printing region equal to the height of the page: writes the p.pg.height member of the PRINTER_PARAMS struct to the p.pg.body.height member.
- asks the user to confirm the resetting of the margins to zero by calling hConfirm with an argument of -SYS_INVALID_MARGINS.
- if the user confirms the resetting of the margins:

  writes zero to each member of prnctrl.tm.

  writes the page height to the p.pg.body.height member of the PRINTER_PARAMS struct.

  writes the page width to the p.pg.body.width member of the PRINTER_PARAMS struct.

sets appropriate text in the *Margins* control.

- returns WN_KEY_NO_CHANGE.

If either the width or the height of the printing region is less than 720 twips:

- asks the user to confirm the resetting of the margins to zero by calling hConfirm with an argument of -SYS_INVALID_MARGINS.

  writes zero to each member of prnctrl.tm.

  writes the page height to the p.pg.body.height member of the PRINTER_PARAMS struct.

  writes the page width to the p.pg.body.width member of the PRINTER_PARAMS struct.

  sets appropriate text in the *Margins* control.

- returns WN_KEY_NO_CHANGE.

Returns WN_KEY_CHANGED.

Otherwise returns WN_KEY_CHANGED.

PAGECTRL

WIN	BWIN	DLGCHAIN	DLGBOX		PAGECTRL
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_key

The PAGECTRL class implements the *Paging control* dialog allowing the user to edit the current page control settings. An example *Paging control dialog* is shown in the following picture:



The Series 3 and Series 3a *Paging control dialog*s are identical apart from the different border style.

The members of the PRINTER_PARAMS struct relevant to the PRNCTRL class are as follows:

`p.pgnum.style` specifies the style used for the page numbering. It also provides an index into the `SYS_PGNO_CHOICE` system resource.

`p.pgnum.offset` specifies the page number offset i.e. the value by which all page numbers are to be offset.

`d.wo_control` specifies whether widows and orphans are allowed and may be either `TRUE` or `FALSE`.

The page control settings are stored in the current `PRINTER` object the handle of which *must* be stored in `w_ws->wserv.printer`.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS pagectrl dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

Property

There is no property associated with the `PAGECTRL` class.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE DIALOG sys_paging_control_dl
    {
    title="Paging control";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_NCEDIT;
            prompt="Number for first page";
            info=NCEDIT
                {
                low=1;
                high=9999;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Allow widows/orphans";
            info=CHLIST{rid=sys_no_yes;};
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Page number style";
            info=CHLIST{rid=sys_pgno_choice;};
            }
        };
    }
```

PAGECTRL methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the dialog using current values stored in the `PRINTER` object.

Obtains the address of a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Sets the content of the *Number for first page* control to the p.pgnum.offset member of the PRINTER_PARAMS struct plus one.

Sets the index of the item selected in the *Allow widows/orphans* control to the d.wo_control member of the PRINTER_PARAMS struct.

Sets the index of the item selected in the *Page number style* control to the p.pgnum.style member of the PRINTER_PARAMS struct.

Returns WN_KEY_CHANGED.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the edited content of the dialog in the PRINTER object.

Obtains the address of a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Senses the value in the *Number for first page* control and writes the result less one to the p.pgnum.offset member of the PRINTER_PARAMS struct.

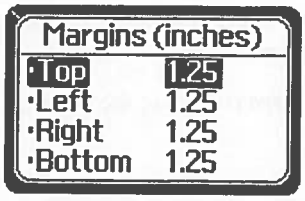
Senses the index of the item selected in the *Allow widows/orphans* control and writes the index to the d.wo_control member of the PRINTER_PARAMS struct.

Senses the index of the item selected in the *Page number style* control and writes the index to the p.pgnum.style member of the PRINTER_PARAMS struct.

MARGINS

WIN	BWIN	DLGCHAIN	DLGBOX		MARGINS
flags		next	item	count	
id			rbuf	current	
			dimrid	underline	
			helprid	absorb	
			flags	changed	
			focus		
~~destroy~~	~~wn_draw~~		destroy	dl_item_replace	dl_dyn_init
wn_calc_position	~~wn_emphasise~~		wn_key	dl_item_append	dl_key
wn_connect			wn_emphasise	dl_init	
wn_dodraw			wn_sense_help	dl_dimmed_message	
~~wn_emphasise~~			wn_set	dl_item_add	
~~wn_key~~			wn_sense	dl_set_size	
wn_position			wn_draw	dl_inq_minsize	
wn_redraw			dl_item_lock	~~dl_dyn_init~~	
~~wn_sense_help~~			dl_item_dim	~~dl_key~~	
wn_visible			dl_set_item_flags		
			dl_set_prompt	*dl_changed*	
~~wn_set~~			dl_take_focus	*dl_focus*	
~~wn_sense~~			dl_handle_to_index	*dl_launch_sub*	
~~wn_draw~~			dl_index_to_handle	*dl_item_new*	
wn_init					

The MARGINS clas implements the *Margins* dialog allowing the user to edit the current page margins. An example *Margins* dialog is shown in the following picture:



The margins in shown in the current printer units.

The page margins are effectively stored in the current PRINTER object the handle of which *must* be stored in `w_ws->wserv.printer`.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS margins dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    }
```

Property

There is no property associated with the MARGINS class.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE DIALOG sys_margins_dl
    {
    title="Margins";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED|DLGBOX_APPEND_UNITS_TITLE;
    controls=
        {
        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Top";
            info=FLTEDIT
                {
                low=0;
                high=9.99;
                ndec=2;
                };
            },

        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Left";
            info=FLTEDIT
                {
                low=0;
                high=9.99;
                ndec=2;
                };
            },

        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Right";
            info=FLTEDIT
                {
                low=0;
                high=9.99;
                ndec=2;
                };
            },
```

```
        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Bottom";
            info=FLTEDIT
                {
                low=0;
                high=9.99;
                ndec=2;
                };
            }
        };
    }
```

MARGINS methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog.

It is assumed that `dlgbox.rbuf` stores the address of a `TMARG` struct defined as follows:

```
typedef struct
    {
    unsigned short int  tmarg;
    unsigned short int  lmarg;
    unsigned short int  rmarg;
    unsigned short int  bmarg;
} TMARG;
```

The significance of the members of the `TMARG` struct is as follows:

`tmarg` specifies the height in twips of the top margin.

`lmarg` specifies the width in twips of the left margin.

`rmarg` specifies the width in twips of the right margin.

`bmarg` specifies the height in twips of the bottom margin.

Converts the margins into the current printer units and then sets them as the content of the dialog controls.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Write the content of the dialog to the `TMARG` struct pointed to by `dlgbox.rbuf`.

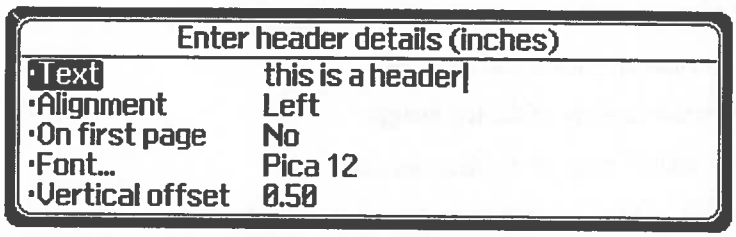
Senses the margins in the dialog controls, converts them into units of twips and then writes the results to the `TMARG` struct pointed to by `dlgbox.rbuf`.

Returns `WN_KEY_CHANGED`.

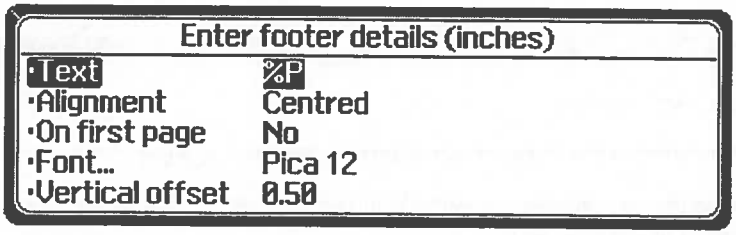
HEADFOOT

WIN	BWIN	DLGCHAIN	DLGBOX		HEADFOOT
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	ph offset
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* ~~dl_launch_sub~~ *dl_item_new*	dl_dyn_init dl_key dl_launch_sub

The HEADFOOT class implements the *Header details* and *Footer details* dialogs allowing the user to edit the the header and the footer respectively. An example *Header details* dialog is shown in the following picture:



An example *Footer details dialog* is shown in the following picture:



The header and the footer are stored in the current PRINTER object the handle of which *must* be stored in `w_ws->wserv.printer`.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS headfoot dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_key
    REPLACE dl_launch_sub
    PROPERTY
        {
        VOID *ph;              points to PAGES_HEADER
        UWORD *offset;          points to positional offset value
        }
    }
```

Note: on the Series 3 the `offset` property is declared as a *signed* word i.e. `WORD`.

Property

`headfoot.ph` a pointer to a `PAGES_HEADER` struct stored by the `PRINTER` object. The `PAGES_HEADER` struct is defined as follows:

```
typedef struct
    {
    SCRLAY_FONT f;
    unsigned char align;
    unsigned char first_page;
    } PAGES_HEADER;
```

the `f` member specifies the default font.

the `align` member contains a value between zero and five inclusive specifying either left, right, centred, justified, two column or three column alignment respectively.

the `first_page` member contains a value which specifies whether or not the header/footer is to appear on the first page. A non-zero value indicates that it is to appear on the first page.

`headfoot.offset` a pointer to a `WORD` stored by the `PRINTER` object. Specifies the vertical offset of the header/footer.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE STRING sys_header_str { str="Enter header details"; }

RESOURCE STRING sys_footer_str { str="Enter footer details"; }

RESOURCE DIALOG sys_headfoot_dl
    {
    title="*";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED|DLGBOX_APPEND_UNITS_TITLE;
    controls=
        {
        CONTROL
            {
            class=C_EDWIN;
            prompt="Text";
            info=EDWIN
                {
                flags=IN_EDWIN_VULEN_CHARACTERS|IN_EDWIN_ACCEPT_TABS;
                maxlen=80;
                vulen=20;
                };
            },

        CONTROL
            {
            class=C_CHLIST;
            prompt="Alignment";
            info=CHLIST{rid=sys_headfoot_align;};
            },
```

```
        CONTROL
            {
            class=C_CHLIST;
            prompt="On first page";
            info=CHLIST{rid=sys_no_yes;};
            },

        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Font ";
            info=TXTMESS
                {
                flags=IN_TEXTWIN_POPOUT;
                };
            },

        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Vertical offset";
            info=FLTEDIT
                {
                low=0;
                high=9.99;
                ndec=2;
                };
            }
        };
    }
```

HEADFOOT methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog.

Note that for a header, `dlgbox.rbuf` should point to an integer value set to `PRINTER_HDR_TOP`, whilst for a footer, it should point to an integer value set to `PRINTER_HDR_BOT`.

Obtains the address of a `PRINTER_PARAMS` struct containing the current printer parameters by sending a `PR_GET_PARAMS` message to `w_ws->wserv.printer`.

If the integer pointed to by `dlgbox.rbuf` is equal to `PRINTER_HDR_TOP`, writes the address of the `p.top` and `p.pg.hdtop` members of the `PRINTER_PARAMS` struct to `headfoot.ph` and `headfoot.offset` respectively and then sets the dialog title from the `SYS_HEADER_STR` system resource.

Otherwise, writes the address of the `p.bot` and `p.pg.hdbot` members of the `PRINTER_PARAMS` struct to `headfoot.ph` and `headfoot.offset` respectively and then sets the dialog title from the `SYS_FOOTER_STR` system resource.

Sets the header/footer text in the *Header/Footer* control: the header/footer text is obtained by sending a `PR_GET_HD` message to `w_ws->wserv.printer`.

Sets the index of the selected item in the *Alignment* control to `headfoot.ph->align`.

Sets the index of the selected item in the *On first page* control to `headfoot.ph->first_page`.

Sets the text in the *Font* control according to the font details in `headfoot.ph.f`.

Converts the value pointed to by `headfoot.offset` to the current printer units and sets this value in the *Vertical offset* control.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the edited content of the dialog in the PRINTER object.

Senses the text in the *Header/Footer* control as appropriate, and then sets this as the header/footer by sending a PR_SET_HD message to w_ws->wserv.printer.

Senses the index of the item selected in the *Alignment* control and then writes the result to headfoot.ph->align.

Senses the index of the item selected in the *On first page* control and then writes the result to headfoot.ph->first_page.

Senses the value in the *Vertical offset* control, converts it from current printer units to twips and then writes the result to the address pointed to by headfoot.offset.

Returns WN_KEY_CHANGED.

DL_LAUNCH_SUB — Launch sub-dialog

```
VOID dl_launch_sub(INT index);
```

Allow the user to edit the current font by launching a *Font selector* dialog.

Opens the current printer resource file by sending a PR_OPEN_WDR message to w_ws->wserv.printer.

Writes appropriate values to a FONTSEL_DATA struct defined as follows:

```
typedef struct
    {
    VOID *wdr;
    SCRLAY_FONT *pf;
    UWORD ret;
    } FONTSEL_DATA;
```

The significance of the members of the FONTSEL_DATA struct is as follows:

wdr — set to the handle of the current WDR object opened by sending a PR_OPEN_WDR message to w_ws->wserv.printer.

pf — the address of headfoot.ph->f i.e. a SCRLAY_FONT struct containing default font data.

ret — set to zero - the *Font selector* dialog sets this to a non-zero value if the font data has been edited

Launches a *Font selector* dialog as described in the *General dialogs* chapter of the *HWIM Reference* manual with as the argument a pointer to the above FONTSEL_DATA struct.

Closes the current printer resource file by sending a PR_CLOSE_WDR message to w_ws->wserv.printer.

Sets the text in the *Font* control according to the font specified by headfoot.ph->f.

PRNMODEL

WIN	BWIN	DLGCHAIN	DLGBOX		PRNMODEL
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	pm pf pi model nsel sf wdrfile
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ wn_init	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ ~~*dl_changed*~~ *dl_focus* ~~*dl_launch_sub*~~ *dl_item_new*	dl_dyn_init dl_key dl_changed dl_launch_sub

The PRNMODEL class implements the *Set printer* dialog allowing the user to select the desired printer and default font. An example *Set printer* dialog is shown in the following picture:



The selected printer and the default font are both stored in the curent PRINTER object the handle of which *must* be stored in `w_ws->wserv.printer`.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS prnmodel dlgbox
    {
    REPLACE dl_key
    REPLACE dl_launch_sub
    REPLACE dl_changed
    REPLACE dl_dyn_init

    TYPES
        {
        typedef struct
            {
            INT path;       entry containing file path in pf
            INT index;      index of printer model info in file
            } MLIST_ITEM;
        }
```

```
    PROPERTY 3
        {
        PR_VASTR  *pm;  handle of object used to store printer models
        PR_VASTR  *pf;  stores names of files where model data is stored
        PR_VAFLAT *pi;  info for each printer model: where stored
        UWORD model;
        WORD nsel;      index of current model in overall list
        SCRLAY_FONT sf;
        TEXT wdrfile[P_FNAMESIZE];
        }
    }
```

Property

`prnmodel.pm` the handle of an instance of the `VASTR` class - this stores the names of the available printer models and provides the data for the *Select printer* control.

`prnmodel.pf` the handle of an instance of the `VASTR` class - this stores the names of the *WDR* files from which the printer models were obtained.

`prnmodel.pi` the handle of an instance of the `VAFLAT` class - this is used to store the index of each printer model in its parent *WDR* files.

The n'th record stores a cross reference for the n'th printer model in the `prnmodel.pm` array.

Each record is organised as an `MLIST_DATA` struct defined as follows:

```
typedef struct
    {
    INT path
    INT index
    } MLIST_ITEM;
```

the `path` member specifies the index of the parent *WDR* file in the `prnmodel.pf` array.

the `index` member specifies the index of the printer model within the *WDR* file.

`prnmodel.model` specifies the model number of the current printer.

`prnmodel.nsel` specifies the index of the current printer model in the `prnmodel.pm` array.

`prnmodel.sf` a `SCRLAY_FONT` struct containing current default font data.

`prnmodel.wdrfile` a buffer containing the name of the current printer resource file.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE DIALOG sys_printer_model
    {
    title="Set printer";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            prompt="Select printer";
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            class=C_CHLIST;
            info=CHLIST{};
            },
```

```
        CONTROL
            {
            class=C_TEXTWIN;
            prompt="Default font ";
            info=TXTMESS
                {
                flags=IN_TEXTWIN_POPOUT;
                };
            }
        };
    }
```

PRNMODEL methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog.

Obtains the address of a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Obtains the current printer model number and the name of the associated *WDR* file by sending a PR_SENSE_MODEL message to w_ws->wserv.printer and writes the results to prnmodel.model and prnmodel.wdrfile respectively.

Writes the d.f member of the PRINTER_PARAMS struct to prnmodel.sf.

Sets suitable text, based on the content of prnmodel.sf, in the *Default font* control.

Creates an instance of the VASTR class and writes its handle to prnmodel.pm. Initialises the VASTR component by sending a VA_INIT message to prnmodel.pm specifying a granularity of 20.

Creates an instance of the VASTR class and writes its handle to prnmodel.pf. Initialises the VASTR component by sending a VA_INIT message to prnmodel.pf specifying a granularity of 32.

Creates an instance of the VAFLAT class and writes its handle to prnmodel.pi. Initialises the VAFLAT component by sending a VA_INIT message to prnmodel.pi specifying records of length equal to the size of an MLIST_ITEM struct and a granularity of four.

Creates an instance of the PMODLOCS class and then sends an LS_SCAN message to the PMODLOCS object: this scans the ROM device and the root and *WDR* directories of all devices on the LOCS:: node for *WDR* printer resource files. This message adds records:

- containing the names of the located *WDR* files: these records are appended to the variable array whose handle is stored in the prnmodel.pf property.
- containing the names of the printer models listed in the located *WDR* file: these records are added to the variable array whose handle is stored in the prnmodel.pm property. Note that on the Series 3a the records are ordered alphabetically and that a duplicate printer model located on a flash pack will replace the ROM version.
- containing a cross reference for each printer model in the *WDR* file: these records are appended to the variable array whose handle is stored in the prnmodel.pi property. The n'th record stores a cross reference for the n'th printer model in the prnmodel.pm array.

Each printer.pi record is organised as an MLIST_ITEM struct.

The MLIST_ITEM struct is defined as follows:

```
typedef struct
    {
    INT path
    INT index
    } MLIST_ITEM;
```

The significance of the members of the MLIST_ITEM struct is as follows:

path specifies the index of the parent *WDR* file in the prnmodel.pf array.

index specifies the index of the printer model within the parent *WDR* file.

Sets the *Select printer* control to display the data in prnmodel.pm array with the current selection set to prnmodel.nsel.

If w_ws->wserv.flags contains PR_WSERV_OWN_DEFAULT_FONT then the method locks the *Default font* control - in consequence the text is visible but may not be edited by the user.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the edited content of the dialog in the PRINTER object.

Obtains the address of a PRINTER_PARAMS struct containing the current printer parameters by sending a PR_GET_PARAMS message to w_ws->wserv.printer.

Senses the index of the item selected in the *Select printer* control and then copies the name of the corresponding printer resource file to prnmodel.wdrfile.

Writes the current printer model number to prnmodel.model and then sets this as the current printer model by sending a PR_SET_MODEL message to w_ws->wserv.printer.

Stores the default font in the PRINTER object by writing prnmodel.sf to the d.f member of the PRINTER_PARAMS struct.

DL_LAUNCH_SUB — Launch sub-dialog

```
VOID dl_launch_sub(INT index);
```

Allow the user to modify the default font characteristics by launching a *Font selector* dialog.

Senses the index of the item selected in the *Select printer* control and then obtains a pointer to the corresponding MLIST_ITEM item in the prnmodel.pi array.

Copies the name of the printer resource file associated with the selected printer model to prnmodel.wdrfile.

Writes the index member of the MLIST_ITEM struct to prnmodel.model.

Creates an instance of the WDR class and then sends a WDR_INIT message specifying prnmodel.wdrfile and prnmodel.model. as the name of the current WDR file and the current printer model number respectively.

Allows the user to modify the default font characteristics stored in prnmodel.sf by launching a *Font selector* dialog: for further details of the *Font selector* dialog see the description of the FONTSEL class in the *General dialogs* chapter of the *HWIM Reference* manual.

Sets appropriate text in the *Default font* control.

DL_CHANGED — Handle item changed messages

```
VOID dl_changed(INT changed);
```

Update the content of the dialog according to the content of the *Select printer* control.

Senses the index of the item selected in the *Select printer* control and then obtains a pointer to the corresponding MLIST_ITEM item in the prnmodel.pi array.

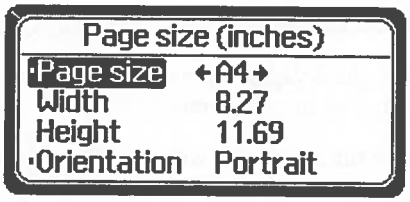
Writes the name of the selected printer resource file to prnmodel.wdrfile and then writes the index member of the MLIST_ITEM struct to prnmodel.model.

Sets appropriate text for the *Default font* control.

PAGESIZE

WIN	BWIN	DLGCHAIN	DLGBOX		PAGESIZE
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ wn_init	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ ~~*dl_changed*~~ *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_changed dl_key

The PAGESIZE class implements the *Page size* dialog allowing the user to specify the desired page size and orientation. An example of a *Page size dialog* is shown in the following picture:



The page size and orientation are stored in the current PRINTER object the handle of which *must* be stored in `w_ws->wserv.printer`.

Class definition

Defined in sub-category file *prntdlgs.cl* (generated header file *prntdlgs.g*).

```
CLASS pagesize dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_changed
    REPLACE dl_key
    }
```

Resources

Defined in the system resource file *s_.rss*.

The following is defined in *hwim.rh*:

```
STRUCT PAGE_SIZE
    {
    WORD width;
    WORD length;
    }
```

The following is defined in *s_.rss*:

```
RESOURCE PAGE_SIZE_ARRAY sys_page_dimensions
    {
    page_size=
        {
        PAGE_SIZE {width=PAGE_WIDTH_A4;        length=PAGE_LENGTH_A4;},
        PAGE_SIZE {width=PAGE_WIDTH_A4;        length=PAGE_LENGTH_A4;},
        PAGE_SIZE {width=PAGE_WIDTH_EXECUTIVE; length=PAGE_LENGTH_EXECUTIVE;},
        PAGE_SIZE {width=PAGE_WIDTH_LEGAL;     length=PAGE_LENGTH_LEGAL;},
        PAGE_SIZE {width=PAGE_WIDTH_LETTER;    length=PAGE_LENGTH_LETTER;},
        PAGE_SIZE {width=PAGE_WIDTH_MONARCH;   length=PAGE_LENGTH_MONARCH;},
        PAGE_SIZE {width=PAGE_WIDTH_DL;        length=PAGE_LENGTH_DL;}
        };
    }
RESOURCE DIALOG sys_pagesize_dl
    {
    title="Page size";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED|DLGBOX_APPEND_UNITS_TITLE;
    controls=
        {
        CONTROL
            {
            class=C_CHLIST;
            flags=DLGBOX_ITEM_NOTIFY_CHANGED;
            prompt="Page size";
            info=CHLIST{rid=sys_page_size;};
            },
        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Width";
            info=FLTEDIT
                {
                low=1;
                high=45;
                ndec=2;
                };
            },
        CONTROL
            {
            class=C_FLTEDIT;
            prompt="Height";
            info=FLTEDIT
                {
                low=1;
                high=45;
                ndec=2;
                };
            },
        CONTROL
            {
            class=C_CHLIST;
            prompt="Orientation";
            info=CHLIST{rid=sys_orient;};
            }
        };
    }
```

PAGESIZE methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog.

Obtains a pointer to a `PRINTER_PARAMS` struct containing the current printer parameters by sending a `PR_GET_PARAMS` message to `w_ws->wserv.printer`.

Sets the index of the item selected in the *Page size* control to the `d.size.choice` member of the `PRINTER_PARAMS` struct.

Not on the Series 3: if `w_ws->wserv.flags` contains `PR_WSERV_METRIC`, sets the upper and lower bounds for the *Width* and *Height* controls to one hundred units and two units respectively.

If the `d.size.choice` member of the `PRINTER_PARAMS` struct is set to the value 1 (corresponding to the second item - the string "Custom" - in the `SYS_PAGE_SIZE` system resource) specifying the custom page size:

- converts the `p.pg.width` member of the `PRINTER_PARAMS` struct from twips to the current printer units, and then sets this as the value in the *Width* control.
- converts the `p.pg.height` member of the `PRINTER_PARAMS` struct from twips to the current printer units, and then sets this as the value in the *Height* control.

Otherwise sets the content of the *Width* and *Height* controls as described for the `DL_CHANGED` message.

If the `p.pdrflags` member of the `PRINTER_PARAMS` struct contains `WDR_PDR_LANDSCAPE`, sets the index of the item selected in the *Orientation* control to one.

Otherwise sets the index of the selected item in the *Orientation* control to zero.

DL_KEY — Handle key input

```
INT dl_key(INT index,INT keycode);
```

Store the selections in the dialog in the `PRINTER` object.

Obtains a pointer to a `PRINTER_PARAMS` struct containing the current printer parameters by sending a `PR_GET_PARAMS` message to `w_ws->wserv.printer`.

Senses the index of the item selected in the *Page size* control and then writes the index to the `d.size_choice` member of the `PRINTER_PARAMS` struct.

Senses the width in the *Width* control, converts the value to the current printer units - i.e. cm or inches - and then writes the value to the `p.pg.width` member of the `PRINTER_PARAMS` struct.

Senses the height in the *Height* control, converts the value to the current printer units - i.e. cm or inches - and then writes the value to the `p.pg.height` member of the `PRINTER_PARAMS` struct.

Senses the index of the item selected in the *Orientation* control and then writes the index to the `p.pdrflags` member of the `PRINTER_PARAMS` struct.

Returns `WN_KEY_CHANGED`.

DL_CHANGED — Handle item changed messages

```
VOID dl_changed(INT changed);
```

Set the content of the *Width* and *Height* controls according to the item selected in the *Page size* control.

The `SYS_PAGE_DIMENSIONS` system resource contains a list of `PAGE_SIZE` structs. The method loads the `PAGE_SIZE` struct having the same index as the item selected in the *Page size* control.

The `PAGE_SIZE` struct is defined as follows:

```
STRUCT PAGE_SIZE
    {
    WORD width;
    WORD length;
    }
```

The significance of the members of the above struct is as follows:

`width` specifies the width of the page in units of twips.

`length` specifies the height of the page in units of twips.

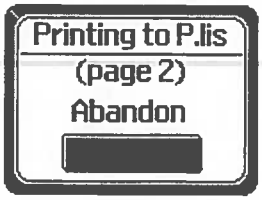
Converts the `width` member of the above struct into the current printer units and then sets the result into the *Width* control.

Converts the `height` member of the above struct into the current printer units and then sets the result into the *Height* control.

PRINTING

WIN	BWIN	DLGCHAIN	DLGBOX		PRINTING
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	pages
destroy wn_calc_position wn_connect wn_dodraw wn_emphasise wn_key wn_position wn_redraw wn_sense_help wn_visible *wn_set* *wn_sense* *wn_draw* *wn_init*	wn_draw wn_emphasise		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize dl_dyn_init dl_key *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_dyn_init dl_set_size printing_set_title printing_do_print printing_done

The `PRINTING` class supports the *Printing* dialog allowing the starting, monitoring and cancelling of printing. An example *Printing* dialog is shown in the following picture:



The handle of the current `PRINTER` object *must* be stored in `w_ws->wserv.printer`.

The `dlgbox.rbuf` property *must* point to an `RBUF_PRINTING` struct which is defined as follows:

```
typedef struct
    {
    PAGES_CALLS calls;
    VOID **ppages;
    } RBUF_PRINTING;
```

The significance of the members of the above struct is as follows:

`calls` a PAGES_CALLS struct which specifies the call backs required by the PAGES object.

The PAGES_CALLS struct is defined as follows:

```
typedef struct
    {
    VOID *hread;
    WORD mread;
    VOID *hdone;
    WORD mdone;
    }
```

the `hread` member specifies the handle of the object that is to receive read call-back messages from the PAGES active object.

the `mread` member specifies the read call-back message that is to be sent by the PAGES active object when it requires the next line of text.

the `hdone` member specifies the handle of the object which is to receive completion call-back messages from the PAGES active object.

the `mdone` member specifies the completion call-back message that is to be sent by the PAGES active object when it has completed either the current page or the entire document.

`ppages` either NULL or a pointer to the handle of the PAGES active object.

(The `hread` and `mread` members of the above PAGES_CALLS struct *must* be set by the owning application.)

Class definition

Defined in sub-category file *printing.cl* (generated header file *printing.g*).

```
CLASS printing dlgbox
    {
    REPLACE dl_dyn_init
    REPLACE dl_set_size
    ADD printing_set_title
    ADD printing_do_print
    ADD printing_done

    CONSTANTS
        {
        PR_WIN_WILL_SKIP        0x4000  Re-use this bit
        }

    TYPES
        {
        typedef struct
            {
            PAGES_CALLS calls;
            VOID **ppages;
            } RBUF_PRINTING;
        }

    PROPERTY 1
        {
        VOID *pages;    pages active object
        }
    }
```

Property

`printing.pages` the handle of an instance of the PAGES active object class - this handles the printing operation. A description of the PAGES class may be found in the *Document Printing Classes* chapter of the *FORM Reference* manual.

Resources

Defined in the system resource file *s_.rss*.

```
RESOURCE DIALOG sys_printing_dialog
    {
    flags=DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD|DLGBOX_ITEM_UNDERLINED;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD;
            info=TXTMESS
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                };
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_ac_abandon;
                };
            }
        };
    }
```

PRINTING methods

DL_DYN_INIT — Dynamic initialisation

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog and start the printing.

Note: `dlgbox.rbuf` *must* point to an appropriately initialised `RBUF_PRINTING` struct.

Sets the text in the *Page number* control - the text is created from the format string in the `SYS_PAGE_IS` system resource and an argument of 9999.

Sets the dialog title by sending `self` a `PRINTING_SET_TITLE` message.

Specifies the completion call-back method by writing `O_PRINTING_DONE` to `dlgbox.rbuf->calls.mdone`.

Specifies the completion call-back handle by writing `self` to `dlgbox.rbuf->calls.hdone`.

Starts the printing by sending `self` a `PRINTING_DO_PRINT` message with as argument the address of `dlgbox.rbuf->calls`.

Writes the handle of the current `PAGES` active object to `printing.pages`.

If `dlgbox.rbuf->ppages` is non-zero, writes `printing.pages` to the address pointed to by `dlgbox.rbuf->ppages`.

DL_SET_SIZE — Set size of dialog

```
VOID dl_set_size(VOID);
```

Set the size of the dialog and the content of the *Page number* control.

Supersends a DL_SET_SIZE message.

Sets the text in the *Page number* control:

- if win.flags contains PR_WIN_WILL_SKIP *and* the printer.p.p.pgbeg member of w_ws->wserv.printer is greater than one, the text is created from the SYS_SKIPPING_PAGE system resource with an argument of one.
- otherwise the text is created from the SYS_PAGE_IS system resource with an argument of one.

The SYS_SKIPPING_PAGE and SYS_PAGE_IS system resources are defined as follows:

```
RESOURCE STRING sys_skipping_page
    {
    str="(skipping page %u)";
    }

RESOURCE STRING sys_printing_to
    {
    str="Printing to %s";
    }
```

PRINTING_SET_TITLE — Set title

```
VOID printing_set_title(VOID);
```

Set the title and the content of the control with index one.

Obtains the name of the printer device - i.e. *Serial*, *Parallel* or a filename - by sending a WS_SENSE_PDEV_TEXT message to w_ws.

Sets the text in the dialog title - the title is created from the SYS_PRINTING_TO system resource and the name of the printer device.

If the printer.p.p.pgbeg member of w_ws->wserv.printer is not equal to one:

- indicates that the first page(s) is (are) to be skipped by setting PR_WIN_WILL_SKIP in win.flags.
- sets the text in the *Page number* control - the text is created from the SYS_SKIPPING_PAGE system resource and an argument of 9999.

PRINTING_DO_PRINT — Start print

```
VOID *printing_do_print(PAGES_CALLS *pcalls);
```

Start the printing operation.

Starts the printing operation by sending a PR_PRINT message to w_ws->printer with an argument of pcalls.

Returns the handle of the PAGES active object which is handling the printing.

PRINTING_DONE — Completion call-back

```
INT printing_done(PAGES_DONE *d);
```

Handle page and document completion.

If the event member of the PAGES_DONE struct pointed to by d is PAGES_DONE_PAGE - indicating that printing of the current page is complete - sets the text in the *Page number* control and then returns FALSE:

- if win.flags contains PR_WIN_WILL_SKIP and the page number of the first page to be printed i.e. w_ws->.printer->printer.p.p.pgbeg is greater than the current page number i.e. d->page, the text is created from the SYS_SKIPPING_PAGE system resource and the current page number.

- otherwise the text is created from the SYS_PAGE_IS system resource and the current page number in d->page.

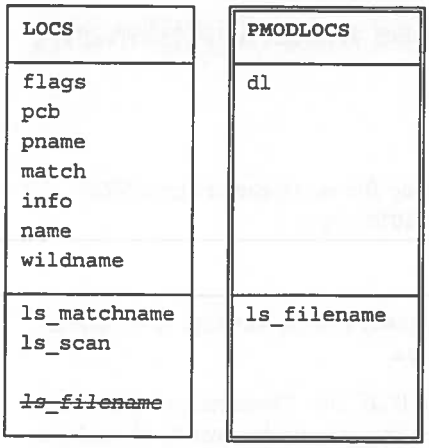
If the event member of the PAGES_DONE struct pointed to by d is PAGES_DONE_DOC - in which case the printing of the document is complete - indicates that no further copies of the document are to be printed by returning FALSE.

If the event member is none of the above - in which case either the printing is complete or an error has occurred while printing - writes NULL to printing.pages, destroys the PRINTING object by sending self a destroy message and then returns FALSE.

For a more detailed explanation of the completion call-back method see the description of the pagedoc_mdone method in the *Formatted Document Content Classes* chapter of the *FORM Reference* manual.

(This method is called by the PAGES active object when it finishes printing the current page/document.)

PMODLOCS

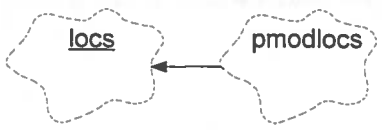


The PMODLOCS class is provided for use by the PRNMODEL class (or a subclass thereof) described in the current chapter. The class may be used to scan the desired devices and directories for *WDR* printer resource files and extract the names of the printer models supported.

The scan adds records:

- containing the names of the located *WDR* files. These records are added to the variable array whose handle is stored in the prnmodel.pf property of the owning class.
- containing the names of the printer models listed in the located *WDR* files. These records are added to the variable array whose handle is stored in the prnmodel.pm property of the owning class.
- containing for each printer model the index of the record storing the name of the printer model and the index of the printer model in the parent *WDR* file. These records are added to the variable array whose handle is stored in the prnmodel.pi property of the owning class.

Class diagram



Class definition

Defined in sub-category file *pmodlocs.cl* (generated header file *pmodlocs.g*).

```
CLASS pmodlocs locs
    {
    REPLACE ls_filename

    PROPERTY
        {
        PR_PRNMODEL *dl;        Owning dialog
        }
    }
```

Property

`pmodlocs.dl` the handle of the object which uses the PMODLOCS component.

PMODLOCS methods

LS_FILENAME — Process matching filename

```
INT ls_filename(TEXT *buf);
```

Process a matching filename.

This method is called when a matching file has been located: the matching file is assumed to be a *WDR* printer resource file the name of which is specified as a zero terminated string by `buf`.

The method adds records:

- containing the name of the located *WDR* file. This record is appended to the variable array whose handle is stored in the `prnmodel.pf` property of the owning class.
- containing the names of the printer models listed in the located *WDR* file. These records are added to the variable array whose handle is stored in the `prnmodel.pm` property of the owning class. Note that on the Series 3a the records are ordered alphabetically and a duplicate printer model located on a flash pack will replace the ROM version.
- containing a cross reference for each printer model in the *WDR* file. These records are appended to the variable array whose handle is stored in the `prnmodel.pi` property of the owning class. The n'th record stores a cross reference for the n'th printer model in the `prnmodel.pm` array. Each record is organised as an MLIST_ITEM struct.

The MLIST_ITEM struct is defined as follows:

```
typedef struct
    {
    INT path
    INT index
    } MLIST_ITEM;
```

The significance of the members of the MLIST_ITEM struct is as follows:

`path` specifies the index of the parent *WDR* file in the `prnmodel.pf` array.

`index` specifies the index of the printer model within the parent *WDR* file.

Returns FALSE.

CHAPTER 17

DIALLING DIALOGS

This chapter documents classes associated with tone dialling. These classes are:

- the HAIDLE class which provides an idle active object especially suited for use with tone dialling.
- the DIALAO class which supports tone dialling and which is used as a component by the DIALDLG class.
- the DIALDLG class which provides a common base for the FDIALDLG and SDIALDLG classes.
- the FREEDIAL class which provides a dialog control that allows free-form tone dialling.
- the FDIALDLG class which provides a dialog that supports free-form tone dialling.
- the CNTRYDLG class which provides a dialog that allows the user to select a country from the list supported by the World database.
- the SDIALDLG class which provides a dialog that allows editing and tone dialling one or more dialling string.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the AIDLE active object class described in the *OLIB Reference* manual.
- the DLGBOX class described in the *Dialog Boxes* chapter of the *HWIM Reference* manual.

HAIDLE

ACTIVE	AIDLE
q priority isactive pcb stat	
destroy ao_init ao_cancel ao_abrun ao_queue ao_run	ao_init ao_run

HAIDLE
rbuf
ao_queue hai_key

The HAIDLE class provides an idle active object, and is used by for example, the DIALAO class described later in this chapter. The HAIDLE class allows the idle active object to be cancelled at any time by pressing the Escape key.

The following illustrates the use of the HAIDLE class:

```
/* compute intensive loop starts here */

WORD stat;

haidle=f_newsend(CAT_DEMO_HWIM,C_HAIDLE,O_AO_INIT);
for (i=0;i<=1000;i++)
    {
    ...
    psend(haidle,AO_QUEUE,&stat);
    ...
    }
```

The AO_QUEUE message returns once all active objects in the queue have run thus ensuring that the compute intensive loop does not monopolise the processor. An example of the use of the HAIDLE class is tone dialling thus ensuring that the tone dialling does not interfere with other tasks and may be cancelled at any moment.

Class diagram



Class definition

Defined in sub-category file *hactive.cl* (generated header file *hactive.g*).

```
CLASS haidle aidle
    {
    REPLACE ao_queue
    ADD hai_key
    PROPERTY
        {
        VOID *rbuf;
        }
    }
```

Property

`haidel.rbuf` used to store the status word for the idle active object.

HAIDLE methods

AO_QUEUE — Start idle object

```
INT ao_queue(VOID *rbuf);
```

Allow queued active objects to run by starting an idle object.

Writes `rbuf` to `haidle.rbuf`.

If the unsigned word pointed to by `rbuf` contains zero, signals the process i/o semaphore - i.e. call `p_iosignal`.

Writes `TRUE` to `active.isactive`.

Directs all future keypresses to its own key method by writing self to `w_ws->wserv.filter` and writing `O_HAI_KEY` to `w_ws->wserv.filmethod`.

Ensures that queued active objects have an opportunity to run by sending an `AM_START` message to `w_am` - this message send only returns once all queued active objects have run.

Writes the original key filter and filter method to `w_ws->wserv.filter` and `w_ws->wserv.filmethod` respectively.

The Series 3a version returns `active.stat`.

The Series 3 version *does not* return a value.

HAI_KEY — Filtered keys handling

```
INT hai_key(INT keycode,INT modifiers);
```

Handle a keypress that may potentially terminate the HAIDLE idle active object.

If keycode is W_KEY_ESCAPE, cancels the idle object as follows:

- indicates that the request was cancelled by writing E_FILE_CANCEL to haidle.rbuf.
- ensures that the last AM_START message send returns by sending an AM_STOP message to w_am.
- indicates that the HAIDLE active object is not active by writing FALSE to active.isactive.

Returns WN_KEY_CHANGED.

DIALAO

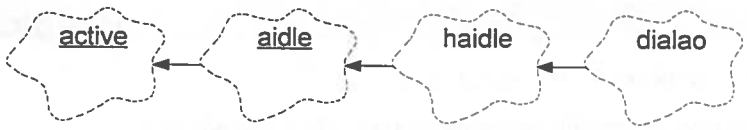
ACTIVE	AIDLE	HAIDLE
q priority isactive pcb stat		rbuf
destroy ~~ao_init~~ ao_cancel ao_abrun ~~ao_queue~~ ~~ao_run~~	ao_init ao_run	~~ao_queue~~ ~~hai_key~~

DIALAO
ao_queue hai_key

The DIALAO class may be used to make an asynchronous request to generate a sequence of dialling tones. The class allows the user to cancel the dialling at any time after the request has been made by pressing the Escape key. Note that the class directs all keypresses to its own hai_key method.

For a description of the sound device see the *I/O Devices Reference* manual.

Class diagram



Class definition

Defined in sub-category file *hactive.cl* (generated header file *hactive.g*).

```
CLASS dialao haidle
Active object supervising dialling
    {
    REPLACE ao_queue
    REPLACE hai_key
    }
```

Property

None.

DIALAO methods

AO_QUEUE — Start dialling object

```
INT dialao_ao_queue(VOID *pcb,TEXT *str,E_DIAL *c);
```

Queue a request to write a tone sequence to the sound channel.

Makes an asynchronous request to the sound channel to play the required tone sequence using `active.stat` as the status word.

The handle of the sound channel should be stored in `pcb` - note that an invalid handle will panic the calling process.

The tone sequence should be stored as a zero terminated string specified by `str`. The timing for the tone dialling is specified by the `E_DIAL` struct pointed to by `c`.

The `E_DIAL` struct is defined as follows:

```
typedef struct
    {
    UBYTE toneLengthTicks;
    UBYTE delayLengthTicks;
    UWORD pauseLengthTicks;
    } E_DIAL;
```

The members of the `E_DIAL` struct have the following significance:

`toneLengthTicks`	the length of a dial tone in units of 1/32 seconds.
`delayLengthTicks`	the time between dial tones in units of 1/32 seconds.
`pauseLengthTicks`	the length of the pause corresponding to the comma and space characters in units of 1/32 seconds.

Further details of the tone dialling may be found in the Sound chapter of the *I/O Devices Reference* manual.

Supersends an `AO_QUEUE` message with an argument of `pcb`.

The Series 3a version returns the value returned by the `AO_QUEUE` message.

The Series 3 version *does not* return a value.

HAI_KEY — Handle filter keys

```
INT dialao_hai_key(INT keycode,INT modifiers);
```

Handle a keypress that may potentially cancel the `DIALAO` idle active object.

If `keycode` is `W_KEY_ESCAPE`:

- cancels any outstanding write request on the sound channel the handle of which is assumed to be in `haidle.rbuf`.
- waits for the cancel request to complete by calling `p_waitstat` with the address of `active.stat` as the argument.
- ensures that the last `AM_START` message send returns by sending an `AM_STOP` message to `w_am`.
- indicates that the `DIALAO` active object is not active by writing `FALSE` to `active.isactive`.

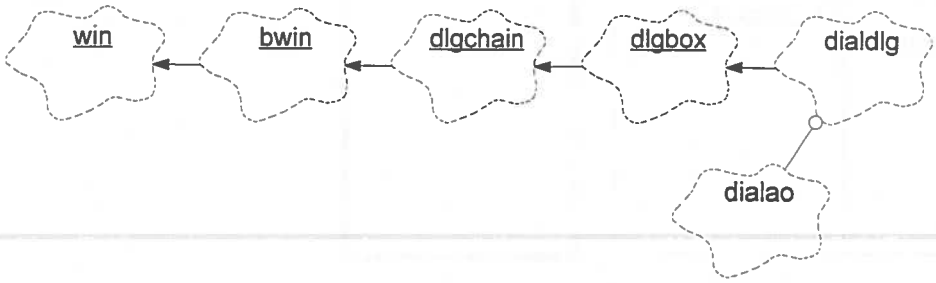
Returns `WN_KEY_CHANGED`.

DIALDLG

WIN	BWIN	DLGCHAIN	DLGBOX		DIALDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	dialao
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~*wn_set*~~ ~~*wn_sense*~~ ~~*wn_draw*~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add ~~*dl_set_size*~~ dl_inq_minsize dl_dyn_init dl_key *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_set_size

The DIALDLG class provides a common base class for the FDIALDLG and SDIALDLG classes. It provides the basic functionality required by a dialog that supports tone dialling.

Class diagram



Class definition

Defined in sub-category file *dialdlgs.cl* (generated header file *dialdlgs.g*).

```
CLASS dialdlg dlgbox
Add active object to appman
    {
    REPLACE dl_set_size
    PROPERTY 1
        {
        VOID *dialao;
        }
    }
```

Property

`dialdlg.dialao` handle of an instance of the DIALAO class - this is used to play a tone sequence.

DIALDLG methods

DL_SET_SIZE — Create a dial active object

```
VOID dl_set_size(VOID);
```

Initialise the help resource and create and initialise a dial active object.

Supersends a DL_SET_SIZE message.

Writes -SYS_HELP_DIALLING to dlgbox.helprid.

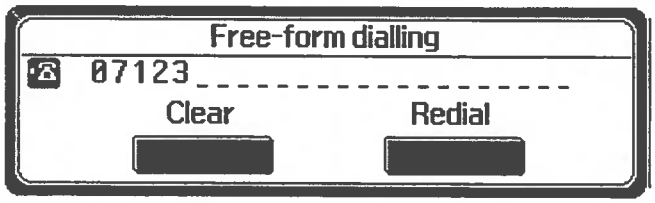
Creates an instance of DIALAO and writes the handle to dialdlg.dialao. Initialises the DIALAO component by sending an AO_INIT message to dialdlg.dialao.

FREEDIAL

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_positio n wn_connect wn_dodraw wn_emphasise ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~destroy~~ ~~wn_init~~ wn_visible lg_draw lg_self_check lg_set_id_pos ~~*lg_sense_width*~~ *lg_update*

FREEDIAL
current str
destroy wn_init wn_draw wn_key lg_sense_width

The FREEFORM class provides a dialog control that supports free-form tone dialling - thus the corresponding tone is generated when a key is pressed. An example of a FREEFORM control used as a dialog component is shown in the following picture:



where the FREEFORM control is the second item in the dialog.

Class diagram



Class definition

Defined in sub-category file *dialdlgs.cl* (generated header file *dialdlgs.g*).

```
CLASS freedial lodger
    {
    REPLACE destroy
    REPLACE wn_init
    REPLACE wn_draw
    REPLACE wn_key
    REPLACE lg_sense_width
    PROPERTY
        {
        WORD current;   /* index of current cursor */
        TEXT str[WR_MAX_DIAL_STRING+2]; /* +2 for ZTS. Max of 24 char */
        }
    }
```

Property

`freedial.current`	the length of the number to tone dial.
`freedial.str`	the number to tone dial - stored as a zero terminated string. See the Sound chapter of the *I/O Device Reference* manual for details of the allowed characters.

FREEDIAL methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the `FREEDIAL` instance.

Ensures that the key click is re-enabled by calling `wDisableKeyClick` with an argument of `FALSE`.

Clears `PR_WSERV_FREEFORM_DIALLING` in `w_ws->wserv.flags` and supersends a `DESTROY` message.

WN_INIT — Initialise freeform dialling

```
VOID wn_init(UBYTE *init,PR_WIN *landlord);
```

Initialise the `FREEDIAL` instance.

Writes an underscore character - '_' - to the first `WR_MAX_DIAL_STRING` elements of `freedial.str`.

Writes zero to `freedial.current`.

Writes `landlord` to `lodger.landlord`.

Sets `PR_WSERV_FREEFORM_DIALLING` in `w_ws->wserv.flags`.

Disables the key click by calling `wDisableKeyClick` with an argument of `TRUE`.

WN_DRAW — Draw dial string

```
VOID wn_draw(VOID);
```

Draw the dial string.

Draws the dial string stored in the first `WR_MAX_DIAL_STRING` elements of `freedial.str`.

The width of the control is taken from the width of the lodger window. The offset of the control is taken from the offset of the lodger window.

The text is drawn in a monospace font with left alignment.

WN_KEY — Add input and dial

```
INT wn_key(INT keycode,INT modifiers);
```

Add keycode to the dial string and play the corresponding tone.

If keycode is greater than or equal to 0x100, returns WN_KEY_NO_CHANGE.

If keycode is neither a comma, an asterisk, a hash - i.e. #, nor a digit:

- converts the character to uppercase.
- if keycode is not in the range 'A' to 'F' inclusive, calls hInfoprint with an argument of -SYS_INVALID_CHAR and returns WN_KEY_NO_CHANGE.

If freedial.current is equal to WR_MAX_DIAL_STRING, resets the dial string by writing an underscore character to the first WR_MAX_DIAL_STRING elements of freedial.str and writing zero to freedial.current.

Writes keycode to freedial.str[freedial.current], increments freedial.current and then draws the control by sending self an LG_DRAW message.

Clears E_SOUND_DISABLE and sets E_SOUND_DEVICE in the current sound flags: see the description of the p_getsnd and p_setsnd routines in the PLIB reference manual for details.

Attempts to open a channel to the sound device (SND:) and on success writes the handle of the sound channel to pcb. On failure restores the sound flags to their original state, calls hInfoprint with an argument of -SYS_SOUND_FAIL and then calls p_leave with an argument of RUN_ACTIVE_CLEANUP_NONOTIFY.

Makes an asynchronous request to the sound channel to play the tone specified by keycode using the timing paramters stored in the D$X environment variable - these are the system wide timing parameters and are stored in the form of an E_DIAL struct. Details of the E_DIAL struct may be found in the Sound chapter of the *I/O Devices Reference* manual.

Closes the sound channel and restores the sound flags to their original state.

Returns WN_KEY_CHANGED.

LG_SENSE_WIDTH — Sense lodger width

```
INT lg_sense_width(VOID);
```

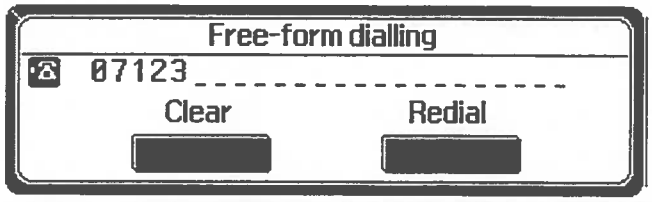
Sense the width in pixels of the lodger window.

Returns WR_MAX_DIAL_STRING multiplied by the maximum width of a character in the system font.

FDIALDLG

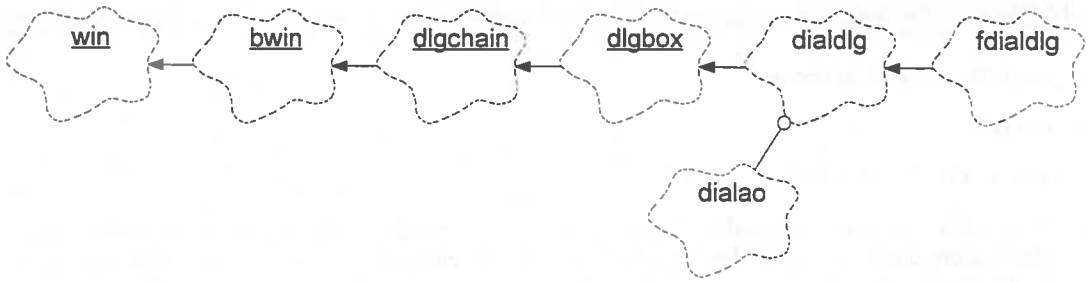
WIN	BWIN	DLGCHAIN	DLGBOX		DIALDLG	FDIALDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	dialao	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add ~~dl_set_size~~ dl_inq_minsize dl_dyn_init ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_set_size	dl_key

The FDIALDLG class provides a dialog that supports free-form tone dialling. An example of a FDIALDLG is shown in the following picture:



where the second item in the dialog is a FREEFORM control as described in the previous section.

Class diagram



Class definition

Defined in sub-category file *dialdlgs.cl* (generated header file *dialdlgs.g*).

```
CLASS fdialdlg  dialdlg
dialog for freeform dial
    {
    REPLACE dl_key
    }
```

Property

None.

Resources

Defined in system resource file *s_rss*.

```
RESOURCE ACLIST_ARRAY sys_freedial_aclist
    {
    button =
        {
        PUSH_BUT
            {
            keycode=W_KEY_DELETE_LEFT;
            str="Clear";
            },
        PUSH_BUT
            {
            keycode=W_KEY_TAB;
            str="Redial";
            }
        };

    }RESOURCE DIALOG sys_freedial_dialog
    {
    title="Free-form dialling";
    flags = DLGBOX_NO_WAIT|DLGBOX_NO_DDP;
    controls=
        {
        CONTROL
            {
            class=C_FREEDIAL;
            prompt=<WS_SYMBOL_PHONE>;
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_freedial_aclist;
                };
            }
        };
    }
```

FDIALDLG methods

DL_KEY — Handle key

```
INT dl_key(INT id,INT keycode);
```

Handle a keypress.

If `keycode` is `W_KEY_DELETE_LEFT`:

- if `freedial.current` is equal to `WR_MAX_DIAL_STRING`, resets the dialling string by writing an underscore character to the first `WR_MAX_DIAL_STRING` elements of `freedial.str` and writing zero to `freedial.current`.
- draws the reset dial string by sending an `LG_UPDATE` message to the `FREEDIAL` item with index one.
- returns `WN_KEY_NO_CHANGE`.

If `keycode` is *not* `W_KEY_TAB`, returns `WN_KEY_NO_CHANGE`.

Clears `E_SOUND_DISABLE` and sets `E_SOUND_DEVICE` in the current sound flags: see the description of the `p_getsnd` and `p_setsnd` routines in the PLIB reference manual for details.

Attempts to open a channel to the sound device and on success writes the handle of the sound channel to `pcb`. On failure restores the sound flags to their original state, calls `hInfoprint` with an argument of `-SYS_SOUND_FAIL` and calls `p_leave` with an argument of `RUN_ACTIVE_CLEANUP_NONOTIFY`.

The dialling string is stored in `dlgbox.item[1].hand->freedial.str` whilst its langth is stored in `dlgbox.item[1].hand->freedial.current`.

If the length of the dialling string is one or zero, makes an asynchronous request to the sound channel to play the dialling sequence.

Otherwise plays the dialling sequence, whilst ensuring that the user may at any time cancel the playing, as follows:

- disables exit and task switch messages by entersending a WS_LOCK message to `w_ws`.
- sends an AO_QUEUE message to `dialdlg.dialao`.
- removes the extra level of locking by sending WS_LOCK message to `w_ws`.

In either case the timing parameters are read from the D$X environment variable - this contains the system timing parameters stored in the form of an E_DIAL struct.

Closes the SND: channel and restores the sound flags to their original state.

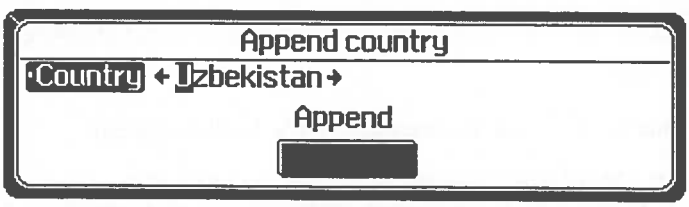
Calls `f_leave`: the argument is either zero or the return value from the WS_LOCK entersend.

Returns WN_KEY_NO_CHANGE.

CNTRYDLG

WIN	BWIN	DLGCHAIN	DLGBOX		CNTRYDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~*wn_set*~~ ~~*wn_sense*~~ ~~*wn_draw*~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message dl_item_add dl_set_size dl_inq_minsize dl_dyn_init ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_key

The CNTRYDLG class allows the user to select a country from the extensive list stored in the World database. An example country selector dialog is shown in the following picture:



The second item in the dialog is a WLDSELWN control which supports selection using the arrow keys or via incremental matching. Further details of the WLDSELWN class may be found in the *Incremental Matchers* chapter of the *HWIM Reference* manual.

Class diagram



Class definition

Defined in sub-category file *dialdlgs.cl* (generated header file *dialdlgs.g*).

```
CLASS cntrydlg dlgbox
    {
    REPLACE dl_key
    }
```

Property

None.

Resources

```
RESOURCE ACLIST_ARRAY sys_append_aclist
    {
    button =
        {
        PUSH_BUT
            {
            keycode=W_KEY_RETURN;
            str="Append";
            }
        };
    }

RESOURCE DIALOG sys_country_selector_dialog
   {
    title="Append country";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_ACTION_LIST|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_WLDSELWN;
            prompt="Country";
            info=WLDSELWN {flags=IN_WLDSELWN_COUNTRY|IN_WLDSELWN_SETHOME; };
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_append_aclist;
                };
            }
        };
    }
```

CNTRYDLG methods

DL_KEY — Handle key

```
INT dl_key(INT id,INT key);
```

Senses the name of the country selected in the WLDSELWN control with index one.

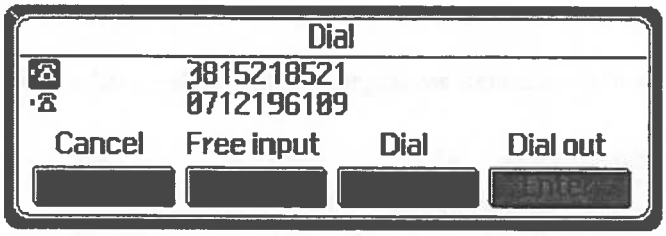
Senses the WLDSELWN dialog control with index one and writes the name of the selected country to `dlgbox.rbuf` - the name is enclosed in square brackets with a leading space asfollows " [Germany]".

Returns WN_KEY_CHANGED.

SDIALDLG

WIN	BWIN	DLGCHAIN	DLGBOX		DIALDLG	SDIALDLG
flags id		next	item rbuf dimrid helprid flags focus	count current underline absorb changed	dialao	
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ *wn_init*	~~wn_draw~~ ~~wn_emphasise~~		destroy wn_key wn_emphasise wn_sense_help wn_set wn_sense wn_draw dl_item_lock dl_item_dim dl_set_item_flags dl_set_prompt dl_take_focus dl_handle_to_index dl_index_to_handle	dl_item_replace dl_item_append dl_init dl_dimmed_message ~~dl_item_add~~ ~~dl_set_size~~ dl_inq_minsize ~~dl_dyn_init~~ ~~dl_key~~ *dl_changed* *dl_focus* *dl_launch_sub* *dl_item_new*	dl_set_size	dl_item_add dl_dyn_init dl_key

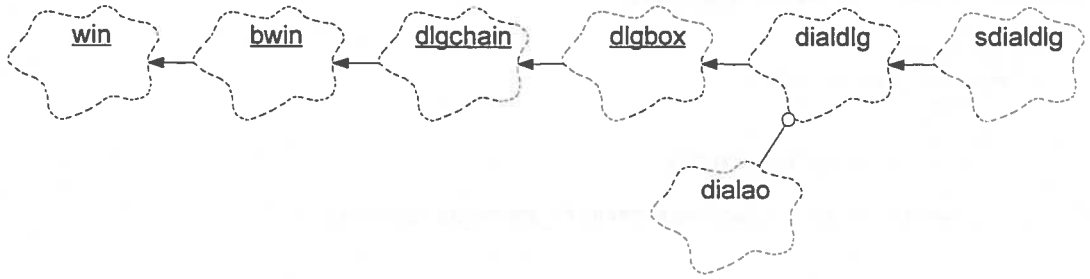
The SDIALDLG class supports tone dialling, editing of dial strings, and free-form tone dialling. An example SDIALDLG dialog is shown in the following picture:



Note that pressing the menu key launches an FDIALDLG dialog as described in an earlier section of this chapter.

On the Series 3a and the Work*about* the *Dial* dialog may contain up to six dial items - i.e. phone numbers - whilst on the Series 3 it may contain up to four dial items.

Class diagram



Class definition

Defined in the sub-category file *dialdlgs.cl* (generated header file *dialdlgs.g*).

```
CLASS sdialdlg  dialdlg
    {
    REPLACE dl_item_add
    REPLACE dl_dyn_init
    REPLACE dl_key
```

```
CONSTANTS
    {
    PAN_TONE_LENGTH_TICKS    4
    PAN_DELAY_LENGTH_TICKS   4
    PAN_PAUSE_LENGTH_TICKS   32
    PC_TONE_LENGTH_TICKS     6
    PC_DELAY_LENGTH_TICKS    4
    PC_PAUSE_LENGTH_TICKS    18
    SMART_DIAL_MAX_PROMPT    11 /* max prompt length */
    }

TYPES
    {
    typedef struct
        {
        TEXT pmt[SMART_DIAL_MAX_PROMPT+1];
        TEXT str[WR_MAX_IN_STRING+2];
        } SMART_DIAL_ITEM;
    typedef struct
        {
        WORD count;          how many separate numbers
        SMART_DIAL_ITEM it[DLGBOX_MAX_ITEM-3];
        } SMART_DIAL_DATA;
    typedef struct
        {
        WORD count;          equals 1
        SMART_DIAL_ITEM it;
        } SMART_DIAL_DATA_S;     uses less stack than SMART_DIAL_DATA
    typedef struct
        {
        UWORD toneLengthTicks;
        UWORD delayLengthTicks;
        UWORD pauseLengthTicks;
        UBYTE dialOutCode[6];        to access an external line
        } DIAL_ENVAR;
    }
}
```

Note: on the Series 3 some of the constants are assigned slightly different values. The Series 3 versions are as follows:

```
PAN_TONE_LENGTH_TICKS    8
PAN_DELAY_LENGTH_TICKS   8
PAN_PAUSE_LENGTH_TICKS   48
```

Property

None.

Resources

Defined in the system resource file *s_rss*.

```
RESOURCE CONTROL sys_smart_dial_item
    {
    class=C_EDWIN;
    prompt=<WS_SYMBOL_PHONE>;
    info=EDWIN
        {
        maxlen=WR_MAX_DIAL_STRING;
        vulen=20;
        flags=IN_EDWIN_VULEN_CHARACTERS|IN_EDWIN_NO_AUTOSELECT;
        };
```

```
RESOURCE ACLIST_ARRAY sys_smart_dial_aclist
    {
    button =
        {
        PUSH_BUT
            {
            keycode=W_KEY_ESCAPE;
            str="Cancel";
            },
        PUSH_BUT
            {
            keycode=W_KEY_MENU;
            str="Free input";
            },
        PUSH_BUT
            {
            keycode=W_KEY_TAB;
            str="Dial";
            },
        PUSH_BUT
            {
            keycode=W_KEY_RETURN;
            str="Dial out";
            }
        };
    }

RESOURCE DIALOG sys_smart_dial_dialog
    {
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_ACTION_LIST|DLGBOX_RBUF_FILLED;
    controls=
        {
        CONTROL
            {
            class=C_TEXTWIN;
            flags=DLGBOX_ITEM_CENTRE|DLGBOX_ITEM_DEAD|DLGBOX_ITEM_UNDERLINED;
            info=TXTMESS;
                {
                flags=IN_TEXTWIN_AL_CENTRE;
                str="Dial";
                };
            },
        CONTROL
            {
            class=C_ACLIST;
            info=ACLIST
                {
                rid=sys_smart_dial_aclist;
                };
            }
        };
    }
```

SDIALDLG methods

DL_ITEM_ADD — Add one or more items to the dialog

```
INT dl_item_add(AD_DLGBOX *par);
```

Add one or more items to the dialog.

If `dlgbox.count` is non-zero, adds an item to the dialog by sending `self` a `DL_ITEM_ADD` message with an argument of `par` - this item *must* specify an appropriate action list. Returns.

If `dlgbox.rbuf->count` is greater than six on the Series 3a or four on the Series 3, then set it equal to either six or four respectively.

If dlgbox.rbuf->count is less than than six on the Series 3a or four on the Series3, adds the first item which *must* specify a title to the dialog by sending self a DL_ITEM_ADD message with an argument of par.

Adds dlgbox.rbuf->count EDWIN controls by sending dlgbox.rbuf->count DL_ITEM_APPEND messages to self with an argument of -SYS_SMART_DIAL_ITEM.

DL_DYN_INIT — Initialise dialog

```
VOID dl_dyn_init(VOID);
```

Initialise the content of the dialog controls.

Note that the dlgbox.rbuf property *must* point to either a SMART_DIAL_DATA struct, or a SMART_DIAL_DATA_S struct followed by zero or more SMART_DIAL_ITEM structs, containing the initialisation data.

The SMART_DIAL_DATA struct is defined as follows:

```
typedef struct
    {
    INT count;
    SMART_DIAL_ITEM it[DLGBOX_MAX_ITEM-3];
    } SMART_DIAL_DATA;
```

The significance of the members of the SMART_DIAL_DATA struct is as follows:

count — the number of smart dial items in the dialog.

it — an array of SMART_DIAL_ITEM structs which stores the data required by the smart dial items.

The SMART_DIAL_ITEM struct is defined as follows:

```
typedef struct
    {
    char pmt[SMART_DIAL_MAX_PROMPT+1];
    char str[WR_MAX_IN_STRING+2];
    } SMART_DIAL_ITEM;
```

The significance of the members of the SMART_DIAL_ITEM struct is as follows:

pmt — the prompt text for the smart dial item - stored as a zero terminated string.

str — the dialling string stored as a zero terminated string.

The dialling string may contain, the digits 0 to 9, upper or lower case alphabetic characters in the range A to F, the characters # (0x23) and * (0x2A) which are converted to F and E respectively and space and comma characters which are interpreted as pauses while dialling.

A space followed by the name of a country enclosed in square brackets e.g. [Germany] may be appended to the dialling string - e.g. 41 35 25 84 [France] - in place of the country code.

Opens a channel to the World database and for each SMART_DIAL_ITEM item in the SMART_DIAL_DATA struct:

- converts the str member into a dialling string using the World database WR_GET_DIAL_STRING service.
- if the conversion succeeds, sets the dialling string into an EDWIN control - the order of the SMART_DIAL_ITEM structs matches the order of the EDWIN controls.
- if a conversion fails, sets the text in the SYS_INVALID_NUM resource into the corresponding EDWIN control and ensures that DLGBOX_ITEM_DEAD is set in the control flags.

DL_KEY — Handle key

```
INT dl_key(INT id,INT keycode);
```

Handle a keypress.

If one of the dialog control holds keyboard focus - i.e. dlgbox.focus is non-zero:

- obtains a dialling string by sensing the content of the EDWIN control with index id.

Otherwise if keycode is either W_KEY_RETURN or W_KEY_TAB:

- calls hInfoPrint with an argument of -SYS_INVALID_NUM and returns WN_KEY_NO_CHANGE.

If keycode is W_KEY_MENU, creates, initialises and makes visible the free-form dialling dialog by sending a WS_FREE_DIAL message to w_ws and then returns WN_KEY_NO_CHANGE.

If keycode is either W_KEY_RETURN or W_KEY_TAB, tone dials the dialling string as follows:

- if keycode is W_KEY_RETURN, prepends the content of the SYS_DIAL_OUT system resource to the dialling string - this defines the dial-out code.
- clears E_SOUND_DISABLE and sets E_SOUND_DEVICE in the current sound flags: see the description of the p_getsnd and p_setsnd routines in the PLIB reference manual for details.
- attempts to open a channel to the sound device and on success writes the handle of the sound channel to pcb. On failure restores the sound flags to their original state, calls hInfoprint with an argument of -SYS_SOUND_FAIL and calls p_leave with an argument of RUN_ACTIVE_CLEANUP_NONOTIFY.
- if the length of the dialling string is one or zero, makes an asynchronous request to the sound channel to play the dialling sequence.
- otherwise, disables exit and task switch messages by entersending a WS_LOCK message to w_ws, plays the tone sequence for the dialling string by sending an AO_QUEUE message to dialdlg.dialao and removes the extra level of locking by sending a WS_LOCK message to w_ws.
- in either case the timing parameters are read from the D$X environment variable - this contains the system timing parameters stored in the form of an E_DIAL struct.
- closes the SND: channel and restores the sound flags to their original state.
- calls f_leave: the argument is either zero or the return value from the WS_LOCK entersend.

If keycode is W_KEY_ESCAPE:

- returns WN_KEY_CHANGED.

CHAPTER 18

HELP CLASSES

This chapter describes the HELPLIST and HELPDLG classes that support the built-in help mechanism: note that these classes are described for interest only since the help mechanism is managed by the system. It is thus unlikely that an application would directly use either class.

The first level of help list may be obtained by simply pressing the Help key and usually contains just a title and a list of help topics as illustrated in the following picture:

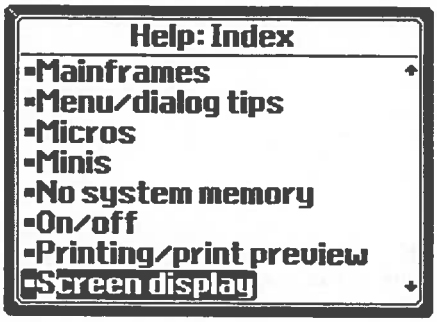


A second level of help list may be obtained from the first level by selecting a topic (with the exception of the *index topic*) and pressing the Return key. The second level of help list usually includes a title and one or more lines of text as illustrated in the following picture:



It may also include a list of topics thus allowing the user to progress to a third level.

An *index help list* is a special form of help list and may be obtained by selecting the *index topic* and pressing Return. It contains a title and a list of system and application help topics. The topics are sorted alphabetically as illustrated in the following example:



The content of the help lists is defined by the help resources in the application's resource file. The ID of the first such resource *must* be written to the `wserv.help_index_id` property of the WSERV object. This may be conveniently done in the `dl_dyn_init` method of the WSERV object as follows:

```
wserv.help_index_id=FIRST_EXAMPLE_HELP;
```

An application can also provide context sensitive help by simply re-setting the `wserv.help_index_id` property as and when required.

The structure of the help resources is perhaps best illustrated by an example. The resources described in the following paragraphs (with one exception) were used to create the help lists shown above.

The content of each help list is specified by means of a HELP_ARRAY resource. The following example is for the first help list illustrated earlier and specifies the title with the topic member, and the list of topics with the topic_id member:

```
RESOURCE HELP_ARRAY first_example_help
    {
    topic="first example";                /* title text */
    topic_id=example_index;    /* id of topic array */
    }
```

A later example also includes some lines of text that appear between the title and the topic list: the lines of text are specified with the strlst member as follows:

```
RESOURCE HELP_ARRAY second_example_help
    {
    topic="second example";                /* title text */
    topic_id=example_index;    /* id of topic array */
    strlst=
        {
        STRING {"One or more lines of text may";},
        STRING {"preceed the bulleted items.";}
        };
    }
```

The topic_id member always contains the ID of a TOPIC_ARRAY resource: the TOPIC_ARRAY that was used for the first example shown earlier is as follows:

```
RESOURCE TOPIC_ARRAY example_index
    {
    id_lst=
        {
        example_main_frames,    /* ids for help screens */
        example_minis,
        example_micros
        }
    }
```

The resource simply contains a list of HELP_ARRAY IDs each of which defines a futher topic help list: the HELP_ARRAY resources for the example TOPIC_ARRAY resource shown above are as follows:

```
RESOURCE HELP_ARRAY example_main_frames
    {
    topic="Mainframes";
    strlst=
        {
        STRING {str="This help screen displays information";},
        STRING {str="about the mainframes topic.";}
        }
    }

RESOURCE HELP_ARRAY example_minis
    {
    topic="Minis";
    strlst=
        {
        STRING {str="This help screen displays information";},
        STRING {str="about the minis topic.";}
        }
    }
```

```
RESOURCE HELP_ARRAY example_micros
    {
    topic="Micros";
    topic_id=example_micros_index;
    strlst=
        {
        STRING {str="This help screen displays information";},
        STRING {str="about the micros topic.";}
        }
    }
```

Usually the help mechanism is used with only two levels of help list. However there is nothing preventing the inclusion of further levels if desired: this is illustrated by the help list defined by the above `example_micros` `HELP_ARRAY` resource which includes a list of topics specified by the `example_micros_index` `TOPIC_ARRAY` as follows:

```
RESOURCE TOPIC_ARRAY example_micros_index
    {
    id_lst=
        {
        example_micros_386,
        example_micros_486
        }
    }
```

The topics in this list are in turn defined by the following three `HELP_ARRAY` resources:

```
RESOURCE HELP_ARRAY example_micros_386
    {
    topic="386 micros";
    strlst=
        {
        STRING {str="Some text describing the";},
        STRING {str="386 processor goes here.";}
        }
    }

RESOURCE HELP_ARRAY example_micros_486
    {
    topic="486 micros";
    strlst=
        {
        STRING {str="Some text describing the";},
        STRING {str="486 processor goes here.";}
        }
    }
```

Precursors

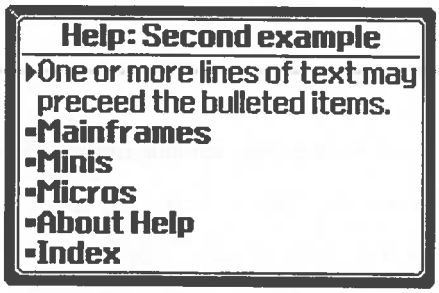
Familiarity with the following topics will aid the understanding of this chapter:

- the `WSERV` object described in the *The WSERV Class* chapter.
- the `WIN` and `BWIN` classes described in the *Windows* chapter.
- the `LISTBOX` class described in the *List Boxes and Menus* chapter.
- the `VAFLAT` and `VASTR` classes described in the *OLIB Reference* manual.
- the `VMATCHER` class described in the *Incremental Matchers* chapter.

HELPLIST

WIN	BWIN	LISTBOX	HELPLIST
flags id		match va flags width matchlen curoff offset current top first last vastart matchstart	list ids firsttop title
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help wn_visible *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ ~~wn_emphasise~~	destroy ~~wn_init~~ ~~wn_key~~ wn_draw wn_emphasise ~~lb_draw_item~~ ~~lb_draw_emphasis~~ lb_size_window lb_item_width lb_take_focus lb_inquire_focus ~~lb_inquire_item~~ lb_inquire_last	wn_init wn_key lb_draw_item lb_draw_emphasis lb_inquire_item lb_item_width

The HELPLIST class may be used to create a help list containing a title, zero or more lines of text and a list of zero or more topics. An example help list is shown in the following picture:

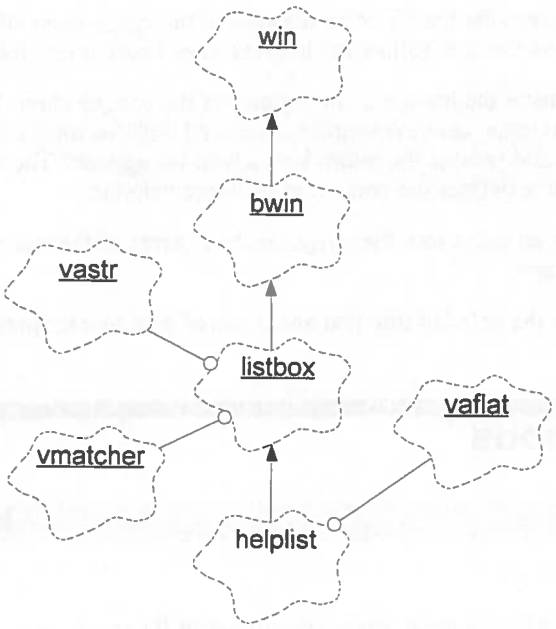


The help list always has a title - in this case "Help: Second example". The title always starts with "Help:".

The help list may include zero or more lines of unbulleted text: these simply provide useful information as illustrated by the second and third lines in the above example.

The help list may also include a list of one or more bulleted topics as illustrated by lines four to eight inclusive in the above example. The user may obtain further information in the form of a help list on a specific topic simply by pressing the Enter key.

Class diagram



Class definition

Defined in sub-category file *help.cl* (generated header file *help.g*).

```
CLASS helplist    listbox
Help topic and topic text list
    {
    REPLACE wn_init
    REPLACE wn_key
    REPLACE lb_draw_item
    REPLACE lb_draw_emphasis
    REPLACE lb_inquire_item
    REPLACE lb_item_width

    TYPES
        {
        typedef struct
            {
            UWORD topic_id;
            TEXT topic[1];  ZTS string followed by UBYTE count of ids
            } HELP_RSC;
        }

    PROPERTY 2
        {
        PR_VASTR *list;          Text for list
        PR_VAFLAT *ids;          Resource ids for list
        UWORD firsttop;          Index of first related topic
        TEXT title[50];
        }
    }
```

Property

`helplist.list`	this stores the handle of an instance of the VASTR class. The records contains the lines of text that follow the help list title. There is one record per line of text.
`helplist.ids`	this stores the handle of an instance of the VAFLAT class. The records contain the IDs of HELP_ARRAY resources associated with the topics. When the user selects a topic and presses the return key, a help list appears. The associated HELP_ARRAY resource defines the content of the topic help list.
`helplist.firsttop`	this is an index into the `helplist.list` array of the record corresponding to the first topic.
`helplist.title`	this is the help list title text and is stored as a zero terminated string.

HELPLIST methods

WN_INIT — Initialise help list

```
VOID wn_init(INT start_id);
```

Initialise the help list specified by the HELP_ARRAY resource with ID `start_id`.

The content and appearance of a help list is specified by means of a HELP_ARRAY resource defined in *hwim.rh* as follows:

```
RESOURCE HELP_ARRAY
    {
    LINK topic_id=0;            /* TOPIC_ARRAY id */
    TEXT topic;                 /* title text */
    LEN BYTE STRUCT strlst[];   /* list of STRINGs */
    }
```

The significance of the members of the HELP_ARRAY resource is as follows:

`topic_id` this member specifies the ID of a TOPIC_ARRAY resource: this resource contains a list of HELP_ARRAY resource IDs each of which defines a topic in the topic list.

This member may be set to SYS_HELP_INDEX_DATA for an *index help list*: i.e a help list containing application *and system* help topics ordered alphabetically which may be selected using incremental matching.

This member may be ignored.

`topic` this member specifies the title text e.g. "Word basics".

`strlst` this member is an array of STRING resources: these define the lines of informative text that appear immediately beneath the title.

This member may be ignored.

The TOPIC_ARRAY resource is defined in *hwim.rh* as follows:

```
RESOURCE TOPIC_ARRAY
    {
    LEN BYTE LINK id_lst[];
    }
```

The significance of the members of the TOPIC_ARRAY resource is as follows:

`id_lst` this member is an array containing one or more HELP_ARRAY resource IDs: each HELP_ARRAY resource defines the content and appearance of a topic help list.

Creates the title from the SYS_HELP_STRING format string resource and the text specified by the `topic` member of the `start_id` resource.

Creates an instance of the VASTR class and writes its handle to `helplist.list`. Iinitialises the VASTR component by sending a VA_INIT message to `helplist.list` specifying a granularity of 16.

Creates an instance of the VAFLAT class and writes its handle to `filelist.ids`. Initialises the VAFLAT instance by sending a VA_INIT message to `helplist.ids` specifying a record length of two and a granularity of sixteen.

If the `topic_id` member of the `start_id` resource is equal to SYS_HELP_INDEX_DATA:

- creates an instance of the VMATCHER class and writes its handle to `listbox.match`. Initialises the VMATCHER component by sending an IM_INIT message with suitable arguments.
- loads the topics specified by the SYS_HELP_INDEX_DATA system resource and then adds records to the `helplist.list` and `helplist.ids` arrays
- loads the topics in the application topic list: the `wserv.help_index_id` property of the WSERV object is assumed to specify the application help resource ID.

Otherwise creates an unsorted help list:

- loads the STRING resources specified by the `start_id` resource and adds the text to the `helplist.list` array.
- loads the topics specified by the `topic_id` member of the `start_id` resource and then adds records to the `helplist.list` and `helplist.ids` arrays.
- if `wserv.flags` does not contain PR_WSERV_BASIC_HELP, add the topics specified by the SYS_HELP_X_HELP system resource. This includes only the *About Help* topic.
- if `wserv.flags` does not contain PR_WSERV_HELP_INDEX, add the topics specified by the SYS_HELP_X_INDEX system resource. This includes only the *Index* topic.

Writes unity to `listbox.current` and `listbox.first`.

Writes an appropriate value to `helplist.first`.

Writes the index of the last item in the help list `helplist.last`. The index of the last item is obtained by sending a VA_COUNT message to `helplist.list`.

Sends an LB_SIZE_WINDOW message to self with an argument of zero.

WN_KEY — Handle keys

```
VOID wn_key(INT keycode,INT modifiers);
```

Handle a keypress.

If `keycode` is W_KEY_RETURN and `listbox.current` is greater than or equal to `helplist.firsttop`, launches a help list for the selected topic by sending a WS_DO_HELP message to the WSERV object.

Otherwise supersends a WN_KEY message with arguments of `keycode` and `modifiers`.

LB_INQUIRE_ITEM — Get item text

```
TEXT *lb_inquire_item(INT index);
```

Return a pointer to the text for the item at line `index`.

If `index` is zero, return the address of the first character in the `helplist.title` property.

Otherwise, return the address of the appropriate record in the `helplist.list` array.

LB_ITEM_WIDTH — Get width of item

```
INT lb_item_width(INT index);
```

Return the width of the help list item at line `index`.

The item width includes the left margin and the width of the text, obtained by sending `self` a LB_INQUIRE_ITEM message.

LB_DRAW_ITEM — Format and draw an item

```
VOID lb_draw_item(TEXT *text,INT index,P_RECT *area);
```

Draw the item with index `index` in the box specified by `area` with text specified by `text`.

If `index` is zero, the text is drawn in a bold font, with centre alignment, and a left margin of width `LISTBOX_OBLOID_INDENT`.

If `index` is greater than or equal to `helplist.firsttop`, the text is drawn in a bold font, with left aligment and a left margin. A bullet character is inserted before the text.

Otherwise, the text is drawn in a system font, with left aligment and a left margin.

LB_DRAW_EMPHASIS — Draw or remove emphasis on item

```
VOID lb_draw_emphasis(INT index,P_RECT *area,INT flag);
```

Emphasise or de-emphasise the item with index `index`.

If `index` is greater than or equal to `helplist.firsttop`, the bullet character and the text are drawn in white on a black background. In this case `area.tl` should specify the top left corner of the item box.

If `index` is less than `helplist.firsttop` and flag is `TRUE`, emphasise the item by drawing a right arrow in the left gutter. Otherwise de-emphasise the item by clearing the right arrow from the left gutter.

In either case `area` should specify the rectangle that encloses the item including the left margin.

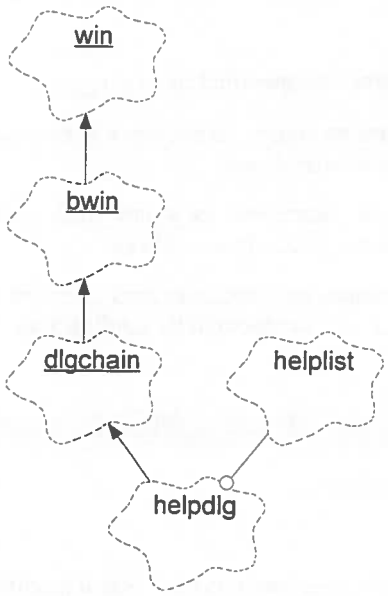
HELPDLG

WIN	BWIN	DLGCHAIN
flags id		next
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw ~~wn_sense_help~~ wn_visible *wn_set* *wn_sense* ~~*wn_draw*~~ ~~*wn_init*~~	~~wn_draw~~ ~~wn_emphasise~~	

HELPDLG
listbox
destroy wn_init wn_key wn_emphasise wn_visible

The description of this class is included for completeness and interest only.

Class diagram



Class definition

Defined in sub-category file *help.cl* (generated header file *help.g*).

```
CLASS helpdlg    dlgchain
Help 'dialog'
    {
    REPLACE destroy
    REPLACE wn_init
    REPLACE wn_key
    REPLACE wn_emphasise
    REPLACE wn_visible
    PROPERTY 1
        {
        PR_HELPLIST *listbox;
        }
    }
```

Property

`helpdlg.listbox` the handle of an instance of the `HELPLIST` class.

HELPDLG methods

DESTROY — Destroy dialog

```
VOID destroy(VOID);
```

Destroy the `HELPDLG` instance.

If `win.flags` contains `HELPDLG_BASIC_HELP`, clears `PR_WSERV_BASIC_HELP` from the `wserv.flags` property of the `WSERV` object. These flags indicate that the highest level of help is the basic help dialog.

Otherwise, if `win.flags` contains `HELPDLG_HELP_INDEX`, clears `PR_WSERV_HELP_INDEX` from the `wserv.flags` property of the `WSERV` object. These flags indicate that the highest level of help is the help index dialog.

If `win.flags` contains `PR_WIN_INITIALISED`, removes the highest level of help dialog by sending a `WS_REMOVE_DIAL` message to the `WSERV` object with `self` as argument, and decrements the `wserv.help` property of the `WSERV` object.

Supersends a `DESTROY` message.

WN_INIT — Initialise help dialog

```
VOID wn_init(INT start_id);
```

Initialise the help dialog whose contents are specified by `start_id`.

If `start_id` is `SYS_HELP_ON_HELP`, sets `PR_WSERV_BASIC_HELP` in the `wserv.flags` property of the `WSERV` object, and sets `HELPDLG_BASIC_HELP` in `win.flags`.

Otherwise, if `start_id` is `SYS_HELP_ON_INDEX`, sets `PR_WSERV_HELP_INDEX` in the `wserv.flags` property of the `WSERV` object, and sets `HELPDLG_HELP_INDEX` in `win.flags`.

Sets `PR_WIN_NO_DDP` in `win.flags`, creates an instance of `HELPLIST` and writes its handle to `helpdlg.listbox`. Initialises the `HELPLIST` component by sending a `WN_INIT` message to `helpdlg.listbox` with an argument of `start_id`.

WN_KEY — Handle keys

```
INT wn_key(INT keycode,INT modifiers);
```

Handle a keypress.

If `keycode` is `W_KEY_ESCAPE`, and `modifiers` contains the Control modifier, destroys all levels of help dialog by repeatedly sending a `DESTROY` message to the `wserv.dial` property of the `WSERV` object until its `wserv.help` property is zero. (See the description of the `WSERV` property for further details).

If `keycode` is `W_KEY_ESCAPE`, and `modifiers` does not contain the Control modifier, destroys the highest level of help dialog by sending a `DESTROY` message to the `WSERV` object `wserv.dial` property.

If `keycode` is `W_KEY_HELP`, and `modifiers` does not contain the Control modifier, adds a help screen for the Help topic by sending a `WS_DO_DIAL` message to the `WSERV` object with an argument of `SYS_HELP_ON_HELP`. The method does nothing if the help screen is already present.

If `keycode` is `W_KEY_HELP`, and `modifiers` contains the Control modifier, adds an index help list by sending a `WS_DO_DIAL` message to the `WSERV` object with an argument of `SYS_HELP_INDEX`. The method does nothing if the index help list is already the highest level.

Otherwise, if `keycode` is none of the above, passes the keypress to the `HELPLIST` component by sending a `WN_KEY` message to `helpdlg.listbox`, passing arguments of `keycode` and `modifiers`.

Returns `WN_KEY_NO_CHANGE`.

WN_EMPHASISE — Emphasise

```
VOID wn_emphasise(UINT flag);
```

Emphasise the help dialog if flag is `TRUE`, otherwise de-emphasise the help dialog.

Sends a `WN_EMPHASISE` message with an argument of `flag` to `helpdlg.listbox`.

WN_VISIBLE — Make the help dialog visible

```
VOID wn_visible(UINT flag);
```

Make the help dialog visible if `flag` is `TRUE`, otherwise make it invisible.

The method simply sends a `WN_VISIBLE` message to `helpdlg.listbox` with `flag` as the argument.

CHAPTER 19

OPL AND COMMS SCRIPT SUPPORT

This chapter describes the following classes:

- the PROGTRAN class which may be used either to translate or to locate an error in a source module.
- the PROGEXEC class which may be used to run a translated source module.
- the PROGFIND class which may be used to locate either of the OPL or Comms script translator modules.

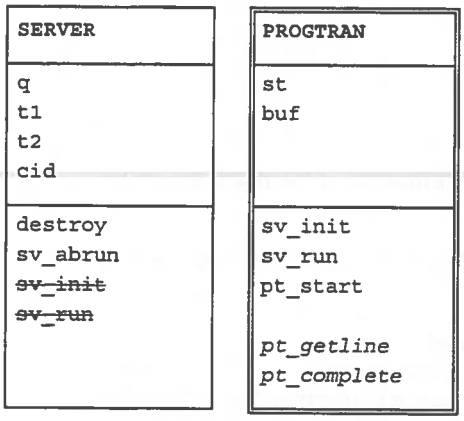
In all cases the source module may contain either OPL code or Comms script commands.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

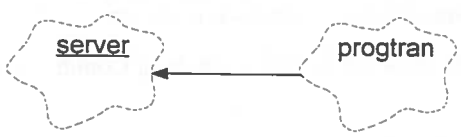
- the ACTIVE class described in the *OLIB Reference* manual.
- the SERVER class described in the *OLIB Reference* manual.

PROGTRAN



The PROGTRAN class may be used to translate either an OPL or a Comms script source module. The class supports both translation of the source file and the location of errors, if they exist.

Class diagram



Class definition

Defined in sub-category file *program.cl* (generated header file *program.g*).

```
CLASS  progtran  server
    {
    REPLACE sv_init          Supersend with parameters
    REPLACE sv_run           Process message
    ADD pt_start             Start translation of a module
    DEFER pt_getline         Get line of source
    DEFER pt_complete        Report completion

    CONSTANTS
        {
        ! Message types
        PT_PROGTRAN_LINE     0x10 Get a line of source
        PT_PROGTRAN_DEATH    0x11 OPL translator death
        }

    TYPES
        {
        typedef struct
            {
            WORD stat;        Completion status
            UWORD mode;       Translation mode
            UWORD line;       Line number
            UWORD offset;     Offset to syntax error
            } PROGTRAN_STATUS;
! Describes the source
        typedef struct
            {
            UWORD pid;                          Process id of source
            TEXT *buffer_offset;                Offset of buffer in source process
            PROGTRAN_STATUS *status_offset;     Offset of status block in source
process
            } PROGTRAN_SOURCE;
        }

    PROPERTY
        {
        PROGTRAN_STATUS st;
        TEXT buf[256];         line buffer
        }
    }
```

Property

`progtran.st` the translator process writes to this property: the significance of the members of the `PROGTRAN_STATUS` struct is as follows:

- `stat` the completion status code: on success this is zero and on failure it is the corresponding error number.
- `mode` the translation mode, which may be one of:
  `FULL_TRANSLATE` for translation of the source module.
  `ERROR_LOCATION` to locate an error in an already translated source module,
  `CHECK_TRANSLATION` to check the source module without translating it.
- `line` the line number of the line of source that contains the error: this member is written to by the translator process on location of an error.
- `offset` the offset into the line of the error: this member is written to by the translator process on location of an error.

`progtran.buf` the full file specification of the OPL module or Comms script file that is to be translated.

PROGTRAN methods

SV_INIT — Initialise

```
VOID sv_init(VOID);
```

Initialise the message server.

Sets PT_PROGTRAN_LINE and PT_PROGTRAN_DEATH as the allowed message types by supersending an SV_INIT message.

SV_RUN — Process message

```
VOID sv_run(MESS *pmsg);
```

Process a message.

If pmsg->type is PT_PROGTRAN_DEATH, frees the message slot with a reply of zero by calling p_mfree. Writes zero to server.cid and then sends self a PT_COMPLETE message. Returns.

If pmsg->type is PT_PROGTRAN_LINE, gets the next line of source by sending self a PT_GETLINE message with pmsg->line as argument. Then frees the message slot by calling p_mfree. The reply sent via this call is the return value from the PT_GETLINE message.

PT_START — Start translation of a module

```
INT pt_start(INT type,TEXT *pname,PROGTRAN_STATUS *pstatus);
```

Start translation of the source module specified by pname. The type argument should be set to either 'O' or 'C' to translate OPL or Comms script source modules respectively. The pstatus argument should point to a progtran.status struct.

Copies the source file specification pointed to by pname into progtran.buf and then obtains the full file specification for the translator program by creating an instance of the PROGFIND class and sending an LS_SCAN message with type as the first argument.

Copies the contents of the PROGTRAN_STATUS struct pointed to by pstatus into progtran.st.

Sets the translator program running in a suspended state by calling p_execc with appropriate command line arguments. Writes the id of the translator process to server.cid and ensures that on abnormal termination of the translator process a PT_PROGTRAN_DEATH message is sent by calling p_logon.

Sets the process running by calling p_presume.

Returns FALSE.

Deferred PROGTRAN methods

PT_GETLINE — Get line of source

```
INT pt_getline(UWORD line);
```

Copy the line of the source file with index line to the buffer with address progtran.buf: the line argument is in the range zero to 32K.

The replacement method should return either the number of bytes copied or E_FILE_EOF if there are no more lines of source.

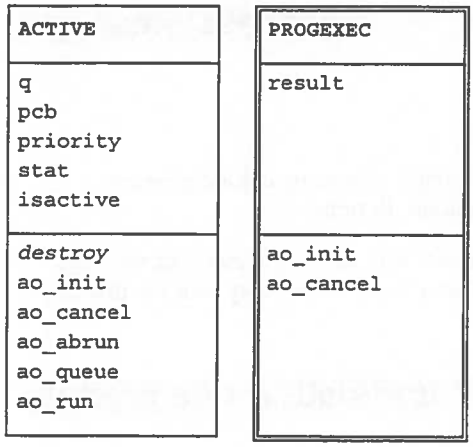
PT_COMPLETE — Report completion

```
VOID pt_complete(VOID);
```

This method is called by the `sv_run` method on abnormal termination of the translator process.

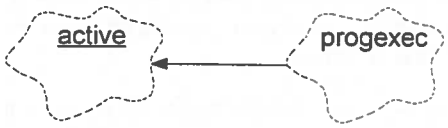
If the translator process is being used to locate an error in the source module, then the `line` and `offset` elements of `progtran.st` give the location of the error.

PROGEXEC



The PROGEXEC class may be used to run either a translated OPL module or a translated Comms script module.

Class diagram



Class definition

Defined in sub-category file *program.cl* (generated header file *program.g*).

```
CLASS  progexec  active
Runs a process of sys$prg?.img, handling inter process communication
Completes when sys$prg? terminates
    {
    REPLACE ao_init            Initialise and queue
    REPLACE ao_cancel        so that destroy is clean

    CONSTANTS
        {
        PROGEXEC_SIGNAL        0x01    signal on completion
        PROGEXEC_NOTIFY        0x02    call notifier on error
        H_COMMAND_TRANSLATE_FILE 'T'
        H_COMMAND_RUN_FILE 'R'
        }
```

```
    TYPES
        {
        ! Used to receive data from sys$prg?.img, on completion
        typedef struct
            {
            WORD  error;                        runtime error
            UWORD line;                         line no of start of proc containing
runtime error
            UWORD offset;                       Q code offset to runtime error
            TEXT err[40];                       runtime error string
            TEXT src[P_FNAMESIZE];              to take source module name
            } PROGEXEC_RESBUF;
        ! Used to pass parameters to sys$prg?.img in p_execc()
        typedef struct
            {
            UWORD  flags;                       mode information for sys$prg?
            UWORD  pid;                         requestor process id
            PROGEXEC_RESBUF *result_offset;     result buffer offset
            } PROGEXEC_PAR;
        }

    PROPERTY
        {
        PROGEXEC_RESBUF result;         to take result
        }
    }
```

Property

`progexec.result` the result buffer that on completion contains details of the run

PROGEXEC methods

AO_INIT — Initialise and queue

```
VOID ao_init(INT type,INT flags,TEXT *pname);
```

Queue for running the translated file specified by `pname`.

The following flags may be passed to the `ao_init` method:

`PROGEXEC_NOTIFY` indicates that the translator is to create an alert or a notify on error

`PROGEXEC_SIGNAL` indicates that the translator is to signal caller of error condition

Copies the filename specified by `pname` to `progtran.buf`.

Obtains the full file specification of the required translator program by creating an instance of the `PROGFIND` class and sending it an `LS_SCAN` message.

Writes `E_GEN_FAIL` to `progexec.result.error`.

Starts the translator running by calling `p_execc` under the protection of `f_leave` with appropriate command line arguments including the address of `progexec.result` and `flags`.

If `flags` contains `PROGEXEC_SIGNAL`:

- writes `PRIORITY_ACTIVE_COMPUTE` to `active.priority`.
- adds `self` to the task queue by sending an `AM_ADD_TASK` message to `w_am`.
- requests asynchronous notification of the abnormal termination of the translator process by calling `p_logona` with the address of `active.stat` as argument.

Otherwise sends `self` a `DESTROY` message.

Sets the translator process in a running state by calling `p_presume`.

AO_CANCEL — Ensure process terminated

```
VOID ao_cancel(VOID);
```

Ensure termination of the translator process.

If `active.isactive` is `TRUE`, writes `FALSE` to `active.isactive`, terminates the translator process by calling `p_pkill` with an argument of `active.pcb` and then waits until the process terminates by calling `p_waitstat` with an argument of `active.stat`.

Write zero to `active.pcb`.

PROGFIND

LOCS
flags pcb pname match info name wildname
ls_matchname ~~ls_scan~~

PROGFIND
pbuf
ls_scan ls_filename

The `PROGFIND` class may be used to locate either:

- the OPL module translator with filename `SYS$PRGO.IMG`.
- the Comms script module translator with filename `SYS$PRGC.IMG`.

In either case the search starts in the `ROM` device and then continues in the `IMG` directory of each node on the `LOC::` device until the required file is located.

Class diagram



Class definition

Defined in sub-category file *program.cl* (generated header file *program.g*).

```
CLASS progfind locs
Find the SYS$PRG?.IMG program translator
    {
    REPLACE ls_scan             Start the scan off
    REPLACE ls_filename         Local/Rom file system name found
    PROPERTY
        {
        UBYTE *pbuf;          Where match name is
        }
    }
```

Property

`progfind.pbuf` the full file specification of the located translator program.

PROGFIND methods

LS_SCAN — Start the scan

```
VOID ls_scan(INT type,UBYTE *pbuf);
```

Search for the translator program specified by `type` and write its full file specification to the buffer at address `pbuf`

A `type` of either 'O' or 'C' indicates that the method is to search for the OPL module translator *SYS$PRGO.IMG* or the Comms script translator *SYS$PRGC.IMG* respectively.

Initialises the result buffer by writing zero to `progfind.pbuf`.

Searches for the required file by sending `self` an `LS_FILENAME` message.

Terminates the scan by sending `self` a `DESTROY` message.

LS_FILENAME — File system name found

```
INT ls_filename(UBYTE *pname);
```

Write the name of the located translator file to the buffer specified by `pname`.

Writes `pname` to `progfind.pbuf`.

Stores the filename in property by copying `P_FNAMESIZE` characters from `locs.name` to `progfind.pbuf`.

Returns `TRUE`.

CHAPTER 20

INCREMENTAL MATCHERS

This chapter describes classes associated with incremental matchers and the World database. These classes are:

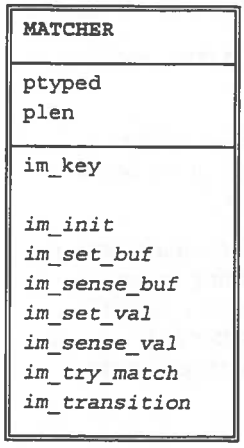
- the MATCHER class which provides a common base class for the VMATCHER and WMATCHER classes.
- the VMATCHER class which supports searching an alphabetically ordered sequence of variable length text records using incremental matching, exact matching and sequential access.
- the WMATCHER class which supports searching the World database for matching cities and countries using incremental matching, exact matching and sequential access.
- the WLDSELWN class which provides a choice list control for accessing information in the World database.

Precursors

Familiarity with the following topics will aid the understanding of this chapter:

- the VAROOT and related classes described in the *Variable Array Classes* chapter of the *OLIB Reference* manual.
- the LODGER class described in the *Windows* chapter of the *HWIM Reference* manual.
- the DLGBOX class described in the *Dialog Boxes* chapter of the *HWIM Reference* manual.
- the Series 3 World application.

MATCHER



The MATCHER class provides a common base class for the VMATCHER and WMATCHER classes described in later sections of this chapter. The class provides the basic functionality for record retrieval via incremental matching, exact matching and sequential access.

Class definition

Defined in sub-category file *matcher.cl* (generated header file *matcher.g*).

```
CLASS   matcher     root
    {
    DEFER im_init
    DEFER im_set_buf
    DEFER im_sense_buf
    DEFER im_set_val
    DEFER im_sense_val
    DEFER im_try_match
    DEFER im_transition
    ADD im_key

    CONSTANTS
        {
        IM_NEW_LEN          1
        IM_NEW_DISPLAY      2
        IM_NO_CHANGE        3
        }
    PROPERTY
        {
        TEXT *ptyped;       Pointer to the typed string
        UBYTE *plen;        Pointer to length byte
        }
    }
```

Property

`matcher.ptyped`	a pointer to the zero terminated match string
`matcher.plen`	a pointer to the length of the match string

MATCHER methods

IM_KEY — Handle key input

```
INT im_key(INT keycode,INT modifiers);
```

Handle a key input that might lead to a new current record.

If `keycode` is `W_KEY_DELETE_LEFT`, removes the last character from the match string at address `matcher.ptyped`, and writes the length of the match string to the length byte at address `matcher.plen`. Locates a matching record by sending an `IM_TRY_MATCH` message to `self` and then returns.

If `keycode` is `W_KEY_ESCAPE`, resets the `MATCHER` instance by replacing the match string with `NULL`, and writing zero to the length byte at address `matcher.plen`. Returns `IM_NEW_LEN`.

If `keycode` is either `W_KEY_RIGHT`, `W_KEY_LEFT`, `W_KEY_HOME` or `W_KEY_END`, resets the `MATCHER` instance by replacing the match string with `NULL`, and writing zero to the length byte at address `matcher.plen`. Sends an `IM_TRANSITION` message to self with `keycode` as the argument. Returns `IM_NEW_DISPLAY`.

Otherwise, if `keycode` is a printable non-control character, adds the character to the end of the match string, and increments the length byte at address `matcher.plen`. Attempts to locate a record matching the new match string by sending an `IM_TRY_MATCH` message to `self`. If the return value from the `IM_TRY_MATCH` message is `IM_NO_CHANGE`, beeps, then restores the match string by removing the last character and decrementing the length byte at address `matcher.plen` and returns `IM_NO_CHANGE`. Otherwise returns the return value from the `IM_TRY_MATCH` message.

Deferred MATCHER methods

IM_INIT — Initialise

```
VOID im_init(UBYTE *plen,UINT maxlen,PR_VAROOT *va);
```

Initialise the MATCHER instance.

Allocate buffer space as required and initialise any property.

IM_SET_BUF — Set current record by match

```
INT im_set_buf(TEXT *str);
```

Set the current record by locating a record that exactly matches the text specified by `str`.

IM_SENSE_BUF — Sense current record data

```
TEXT *im_sense_buf(VOID);
```

Return the address of the current record.

IM_SET_VAL — Set current record

```
VOID im_set_val(UINT index);
```

Set the current record by index: the records are assumed to be uniquely indexed.

IM_SENSE_VAL — Sense current record index

```
INT im_sense_val(VOID);
```

Return the index of the current record.

IM_TRY_MATCH — Try clever match

```
INT im_try_match(VOID);
```

Locate a record that matches the match string at `matcher.ptyped`.

The subclass would normally allow incremental matching whereby the first record whose initial text matches that at `matcher.ptyped` would be selected. The method returns the index of the matching record.

IM_TRANSITION — Handle transition

```
VOID im_transition(UINT type);
```

Handle a keypress that might lead to the selection of a new current record.

The subclass may wish to add extra functionality such as limiting the range of allowed records, adding an offset from which the matching commences, or restricting records to those terminating with the `.pic` extension, for example.

VMATCHER

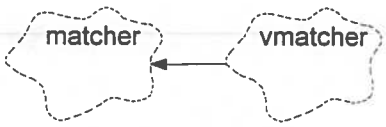
MATCHER
ptyped plen
im_key ~~im_init~~ ~~im_set_buf~~ ~~im_sense_buf~~ ~~im_set_val~~ ~~im_sense_val~~ ~~im_try_match~~ ~~im_transition~~

VMATCHER
va index first last txtoff
destroy im_init im_set_buf im_sense_buf im_set_val im_sense_val im_try_match im_transition im_set_range

The VMATCHER class may be used to search an alphabetically ordered sequence of variable length text records. Incremental matching, exact matching and sequential access are supported.

Examples of the use of the VMATCHER class may be found in the descriptions of the FNEDIT and CHLIST classes.

Class diagram



Class definition

Defined in sub-category file *matcher.cl* (generated header file *matcher.g*).

```
CLASS   vmatcher    matcher
Variable array matcher
    {
    REPLACE destroy
    REPLACE im_init
    REPLACE im_set_buf
    REPLACE im_sense_buf
    REPLACE im_set_val
    REPLACE im_sense_val
    REPLACE im_try_match
    REPLACE im_transition
    ADD im_set_range

    PROPERTY
        {
        PR_VAROOT *va;  Handle of associated variable array
        UWORD index;    Index into array of current match
        UWORD first;    Index of first item in array to match
        UWORD last;     Index of last item in array to match (0 = VA_COUNT-1)
        UWORD txtoff;   Offset within buffer returned by pbuf to text
        }
    }
```

Property

`vmatcher.va`	the handle of an array of variable length text records
`vmatcher.index`	the index of the current record
`vmatcher.first`	the index of the first record in the search range

`vmatcher.last` the index of the last record in the search range: a zero index indicates that the search is to extend to the last record in the array.

`vmatcher.txtoff` an offset into each record from which the matching should commence.

VMATCHER methods

DESTROY — Destroy

```
VOID destroy(PR_VMATCHER *self);
```

Free the match string buffer at address `matcher.ptyped` and supersend a `DESTROY` message.

IM_INIT — Initialise

```
VOID im_init(UBYTE *plen,UINT maxlen,PR_VAROOT *va);
```

Initialise the instance of `VMATCHER`.

Initialises the records by writing `va` to `vmatcher.va`, allocates at least `maxlen` bytes for the match string and writes the address of the first byte to `matcher.ptyped`.

IM_SET_BUF — Set current record by match

```
INT im_set_buf(TEXT *str);
```

Locate the first record that exactly matches the text specified by `str` excluding records outside of the current range: see the description of the `im_set_range` method for details of the range.

If a matching record is located, writes its index to `vmatcher.index` and return zero.

Otherwise return -1.

IM_SENSE_BUF — Sense current record data

```
TEXT *im_sense_buf(VOID);
```

Return the address of the current record data offset by `vmatcher.txtoff`.

Obtains the address of the current record data by sending a `VA_PBUF` message to `vmatcher.va` with an argument of `vmatcher.index`.

IM_SET_VAL — Set current record by index

```
VOID im_set_val(UINT index);
```

Set the current record by writing `index` to `vmatcher.index`.

IM_SENSE_VAL — Sense current record

```
INT im_sense_val(VOID);
```

Sense the index of the current record: the method does no more than return `vmatcher.index`.

IM_TRY_MATCH — Try intelligent match

```
INT im_try_match(VOID);
```

Locate the first record that matches the text at address `matcher.ptyped` excluding records outside of the current range. See the description of the `im_set_range` method for details of the current range. The matching is case insensitive.

Searches the records to locate the first one that matches the text at address `matcher.ptyped`. A matching record is found when the first `len` characters match those stored at address `matcher.ptyped`. The length `len` is read from address `matcher.plen`.

If there is no matching record, returns `IM_NO_CHANGE`.

If the matching record is the current record, and thus has index `vmatcher.index`, returns `IM_NEW_LEN`.

Otherwise if the matching record is not the current record, makes it the current record by writing its index to `vmatcher.index`, and returns `IM_NEW_DISPLAY`.

IM_TRANSITION — Handle transition

```
VOID im_transition(UINT keycode);
```

Handle a keypress that might lead to the selection of a new current record. Only records in the range are considered: see the description of the `im_set_range` method for details of the range.

If `vmatcher.last` is invalid as it is both non-zero and less than `vmatcher.first`, returns.

If `keycode` is either `W_KEY_RIGHT`, or `W_KEY_DOWN`, and the last record has not been reached, moves to the next record by incrementing `vmatcher.index`.

If `keycode` is `W_KEY_HOME`, moves to the first record by writing `vmatcher.first` to `vmatcher.index`.

If `keycode` is `W_KEY_END`, moves to the last record by writing its index to `vmatcher.index`.

If type is either `W_KEY_LEFT`, or `W_KEY_UP`, and the first record has not been reached, moves to the previous record by decrementing `vmatcher.index`.

IM_SET_RANGE — Set range

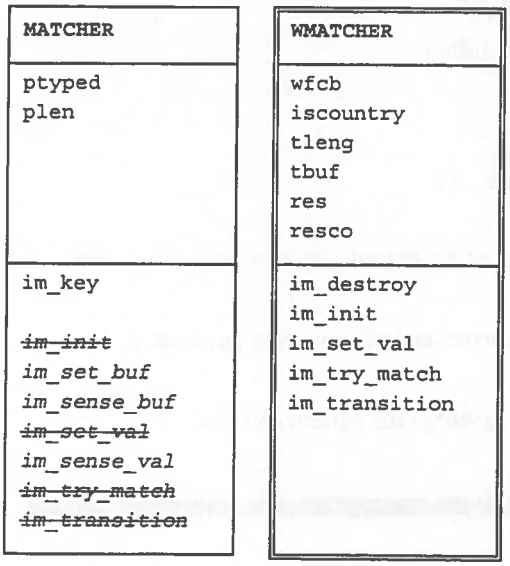
```
VOID im_set_range(UWORD first,UWORD last,UWORD txtoff);
```

Set the record range to start at the record with index `first` and to extend to either the record with index `last` or the last record in the array. Ensure that the matching starts at an offset into each record of `txtoff`, i.e. the first `txtoff` characters are skipped when matching.

Writes `first` to `vmatcher.first`, writes `last` to `vmatcher.last` and writes `txtoff` to `vmatcher.txtoff`.

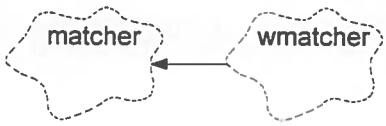
The record range extends either to the record with index `vmatcher.last`, or to the last record in the array if `vmatcher.last` is zero.

WMATCHER



The WMATCHER class may be used to search the World database for matching cities or countries. Incremental and exact matching are supported as well as sequential access. An example of the use of the WMATCHER class is provided by the WLDSELWN class described in the following section of this chapter. Note that the WMATCHER class does not support data retrieval: it simply provides a means of navigating the World database.

Class diagram



Class definition

Defined in sub-category file *wldselwn.cl* (generated header file *wldselwn.g*).

```
CLASS   wmatcher    matcher
    {
    REPLACE destroy
    REPLACE im_init
    REPLACE im_set_val
    REPLACE im_try_match
    REPLACE im_transition

    CONSTANTS
        {
        GEOG_MAX_NAME   22      Actually 20 but room needed for trailing zero
        PR_WMATCHER_INDETERMINATE       0x02    Neither city nor country
        }

    PROPERTY
        {
        VOID *wfcb;                     Control block for world
        UBYTE iscountry;                TRUE for country mode
        UBYTE tleng;                    The current typed length
        TEXT tbuf[GEOG_MAX_NAME];       The current typed string
        WR_FIND_RES res;                Holds latest result
        WORD resco;                     TRUE if restricted
        }
    }
```

Property

`wmatcher.wfcb` — the handle of a channel to the World database: for details of the World database see the *World* chapter of the *I/O Devices Reference* manual.

wmatcher.iscountry	TRUE to access countries: FALSE to access cities
wmatcher.tleng	the current length of the match string: this must not exceed WR_MAX_NAME
wmatcher.tbuf	a buffer holding the current match string

wmatcher.res	The WR_FIND_RES struct is defined as follows:
	`typedef struct`
	`{`
	`TEXT city[WR_MAX_NAME+1];`
	`TEXT country[WR_MAX_NAME+1];`
	`} WR_FIND_RES;`
	The city member specifies the name of the current city as a zero terminated string.
	The country member specifies the current country as a zero terminated string.

wmatcher.resco	TRUE if the search is restricted to cities within the current country

WMATCHER methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the WMATCHER instance by closing the i/o channel specified in wmatcher.wfcb and supersending a destroy message.

IM_INIT — Initialise

```
VOID im_init(VOID);
```

Initialise the WMATCHER instance.

Writes the address of the match string buffer to matcher.ptyped and writes the address of wmatcher.tleng to matcher.plen.

Opens an i/o channel to the World database and writes the handle of the channel to wmatcher.wfcb. Sets the current record to the first city in the database and writes the names of the city and the country in which the city resides to wmatcher.res.

IM_SET_VAL — Set current record

```
VOID im_set_val(WR_FIND_RES *match);
```

Set the current record by an exact match to the city, or if this is NULL, the country specified in the WR_FIND_RES struct with address match. The matching is not case sensitive.

The WR_FIND_RES struct is defined as follows:

```
typedef struct
    {
    TEXT city[WR_MAX_NAME+1];
    TEXT country[WR_MAX_NAME+1];
    } WR_FIND_RES;
```

The members of the WR_FIND_RES struct have the following significance:

city	either NULL or the name of the target city specified as a zero terminated string.
country	either NULL or the name of the target country specified as a zero terminated string.

If a matching city is found, writes the names of the city and country in which the city resides to wmatcher.res and returns IM_NEW_DISPLAY.

If a matching country is found, writes the names of the country and its capital city to `wmatcher.res` and returns `IM_NEW_DISPLAY`.

If no matching record is found, returns `IM_NO_CHANGE`.

IM_TRY_MATCH — Try intelligent match

```
INT im_try_match(VOID);
```

Set the current record to the first record that matches the current match string. The matching is not case sensitive.

If `wmatcher.tleng` is greater than `WR_MAX_NAME` returns `IM_NO_CHANGE`.

If `wmatcher.iscountry` is `TRUE`, searches for a matching country, writes the names of the country and its capital city to `wmatcher.res` and returns `IM_NEW_DISPLAY`.

If `wmatcher.iscountry` is `FALSE`, searches for a matching city, writes the names of the city and the country in which the city resides to `wmatcher.res` and returns `IM_NEW_DISPLAY`.

If no match is found, returns `IM_NO_CHANGE`.

IM_TRANSITION — Handle transition

```
VOID im_transition(UINT keycode);
```

Handle the keypress specified by `keycode`.

If `wmatcher.resco` is `TRUE`, sets the current record to the next city in the current country and returns.

If `keycode` is `W_KEY_RIGHT`, and `wmatcher.iscountry` is `TRUE`, sets the current record to the next country.

If `keycode` is `W_KEY_RIGHT`, and `wmatcher.iscountry` is `FALSE`, sets the current record to the next city.

If `keycode` is `W_KEY_LEFT`, and `wmatcher.iscountry` is `TRUE`, sets the current record to the previous country and returns.

If `keycode` is `W_KEY_LEFT`, and `wmatcher.iscountry` is `FALSE`, sets the current record to the previous city and returns.

If `keycode` is `W_KEY_HOME`, sends an `IM_TRY_MATCH` message to `self` and returns.

Otherwise, sends an `IM_TRY_MATCH` message to `self`. If `wmatcher.country` is `TRUE`, sets the current record to the previous country. If `wmatcher.country` is `FALSE`, sets the current record to the previous city.

WLDSELWN

WIN	LODGER
flags id	landlord offset width
~~destroy~~ wn_calc_position wn_connect wn_dodraw ~~wn_emphasise~~ ~~wn_key~~ wn_position wn_redraw wn_sense_help ~~wn_visible~~ ~~wn_set~~ ~~wn_sense~~ ~~wn_draw~~ ~~wn_init~~	~~destroy~~ ~~wn_init~~ wn_visible lg_draw lg_self_check lg_set_id_pos ~~lg_sense_width~~ *lg_update*

WLDSELWN
mat tleng flags other
destroy wn_init wn_draw wn_set wn_key wn_sense wn_emphasise lg_sense_width wld_restrict

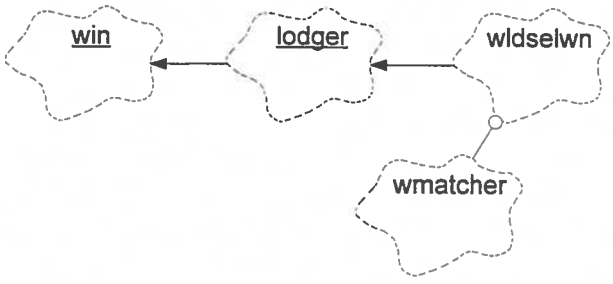
The WLDSELWN class provides for the retrieval of information from the World database and supports sequential access and incremental matching via the keyboard. An instance of the WLDSELWN class may be used as a control in a lodger window.

An example of the WLDSELWN class being used as a control in a dialog is shown in the following picture:



In the above picture the controls are coupled in that the current city always resides in the current country. The current city could also be restricted to cities in the current country. This is known as *restriction*.

Class diagram



Class definition

Defined in sub-category file *wldselwn.cl* (generated header file *wldselwn.g*).

```
CLASS   wldselwn   lodger
    {
    REPLACE destroy
    REPLACE wn_init
    REPLACE wn_draw             Draw whole city/country name
    REPLACE wn_set              Set current city/country to str
    REPLACE wn_key
    REPLACE wn_sense
    REPLACE wn_emphasise
    REPLACE lg_sense_width      Returns max width
    ADD wld_restrict
```

```
CONSTANTS
    {
    PR_WLDSELWN_COUNTRY             0x01
    PR_WLDSELWN_FIRST_PARTNER       0x02
    PR_WLDSELWN_NOT_IN_DIALOG       0x04
    PR_WLDSELWN_RESTRICTED          0x08

    IN_WLDSELWN_CITY                0
    IN_WLDSELWN_COUNTRY             PR_WLDSELWN_COUNTRY
    IN_WLDSELWN_FIRST_PARTNER       PR_WLDSELWN_FIRST_PARTNER
    IN_WLDSELWN_NOT_IN_DIALOG       PR_WLDSELWN_NOT_IN_DIALOG
    IN_WLDSELWN_SECOND_PARTNER      0x10
    IN_WLDSELWN_SETHOME             0x20
    IN_WLDSELWN_SETDFLT             0x40
    }

TYPES
    {
    typedef struct
        {
        UWORD flags;
        PR_WIN *other;
        } IN_WLDSELWN;
    typedef struct
        {
        VOID *wfcb;          io channel
        WR_EXTRA_DATA data;
        } SENSE_WLDSELWN;          use WR_FIND_RES for wn_set
    }

PROPERTY
    {
    PR_WMATCHER *mat;
    UBYTE *tleng;
    UWORD flags;
    PR_WIN *other;      If non-NULL, then one of a pair
    }
}
```

Property

`wldselwn.mat` the handle of an instance of WMATCHER.

`wldselwn.tleng` a pointer to the current length of the match string.

`wldselwn.flags` an ORed combination of flags that determine the behaviour of the component: see the description below.

`wldselwn.other` the handle of a second instance of WLDSELWN or NULL.

The initial behaviour and content of a WLDSELWN control is specified by ORing a suitable combination of the following flags into the `flags` member of the WLDSELWN resource struct:

`IN_WLDSELWN_FIRST_PARTNER` specifies that the control is the first partner in a coupled pair. If the control is in a dialog, then it must be placed immediately above the second control.

`IN_WLDSELWN_SECOND_PARTNER` specifies that the control is the second partner in a coupled pair. If the control is in a dialog, then it must be placed immediately below the first control.

`IN_WLDSELWN_NOT_IN_DIALOG` specifies that the control is a component in a non-dialog lodger window

`IN_WLDSELWN_SETDFLT` specifies that the control is to display the default country or city as appropriate. This flag must not be set for the first partner in a pair: setting this flag for the second partner is sufficient.

`IN_WLDSELWN_SETHOME` specifies that the control is to display the home city or country as appropriate. This flag must not be set for the first partner in a pair: setting this flag for the second partner is sufficient.

WLDSELWN methods

DESTROY — Destroy

```
VOID destroy(VOID);
```

Destroy the WLDSELWN component.

If wldselwn.flags does not contain IN_WLDSELWN_FIRST_PARTNER, and wldselwn.mat is non-zero, destroys the WMATCHER component by sending a DESTROY message to wldselwn.mat.

Supersend a DESTROY message.

WN_INIT — Initialise

```
VOID wn_init(IN_WLDSELWN *par,PR_WIN *landlord);
```

Initialise the control according to the content of the IN_WLDSELWN struct pointed to by par.

Write landlord to lodger.landlord and write par->flags to wldselwn.flags.

If the control is either an isolated WLDSELWN control, or the second partner in a coupled pair, creates an instance of WMATCHER and writes its handle to wldselwn.mat. Initialises the WMATCHER component by sending an IM_INIT message to wldselwn.mat and writes the matcher.plen property of the WMATCHER instance to wldselwn.tleng.

If the control is the second partner in a pair, writes the handle of the first partner to wldselwn.other. The handle of the other control is either copied from par->other, in which case the control is not a dialog component, or obtained via the sending of DL_HANDLE_TO_INDEX and DL_INDEX_TO_HANDLE messages to the dialog. Writes the handle of the WMATCHER component to the wldselwn.mat property of both controls. Writes the address of wldselwn.tleng to the wldselwn.tleng property of the first partner.

If par->flags contains IN_WLDSELWN_SETHOME, sets the current record to the home city, or country, according to the content of wldselwn.flags, and draws the control by sending a WN_DRAW message to self.

If par->flags contains IN_WLDSELWN_SETDFLT, sets the current record to the default city, or country, according to the content of wldselwn.flags, and draws the control by sending a WN_DRAW message to self.

WN_DRAW — Draw current record

```
VOID wn_draw(VOID);
```

Draw the control.

Displays the name of the current record in the system font and, if win.flags contains PR_WIN_EMPHASISED, emphasizes the control. The current record is the name of the current country if wldselwn.flags contains PR_WLDSELWN_COUNTRY, otherwise it is the name of the current city.

If wldselwn.flags contains both IN_WLDSELWN_NOT_IN_DIALOG and PR_WLDSELWN_COUNTRY and the control is one of a pair the other of which has PR_WLDSELWN_RESTRICTED set in its wldselwn.flags property, then the method draws the text in the SRT_SPACE_ONLY system resource after the name of the country.

On English language machines the SRT_SPACE_ONLY system resource is defined as follows:

```
RESOURCE STRING srt_space_only
    {
    str=" (only)";
    }
```

WN_SET — Set current record

```
VOID wn_set(WR_FIND_RES *pset);
```

Set the current record by an exact match to the text specified in the WR_FIND_RES struct pointed to by pset.

The WR_FIND_RES struct is defined as follows:

```
typedef struct
    {
    TEXT city[WR_MAX_NAME+1];
    TEXT country[WR_MAX_NAME+1];
    } WR_FIND_RES;
```

The members of the WR_FIND_RES struct have the following significance:

city — either NULL or the name of the target city specified as a zero terminated string.

country — either NULL or the name of the target country specified as a zero terminated string.

Sets the current record by sending an IM_SET_VAL message to wldselwn.mat passing as argument pset. A NULL city should be specified when setting the country.

Resets the current match string by writing the NULL string to the matcher.ptyped property of wldselwn.mat and zero to the matcher.plen property of wldselwn.mat.

Sets PR_WMATCHER_INDETERMINATE in the wmatcher.iscountry property of the WMATCHER component.

Draws the control by sending an LG_DRAW message to self, and if the control is one of a pair, draws the other control by sending an LG_DRAW message to wldselwn.other.

WN_KEY — Handle key input

```
INT wn_key(UINT keycode,UINT modifiers);
```

Handle a keypress that might lead to the selection of a new current record.

Sends an IM_KEY message to wldselwn.mat passing keycode and modifiers as arguments: the WMATCHER component is primed to ensure that it matches either by country if wldselwn.flags contains IN_FNSELWN_COUNTRY, or by city if wldselwn.flags contains IN_FNSELWN_CITY.

If the return value is IM_NEW_LEN, draws the cursor one character to the right of its previous position and returns IM_NEW_LEN.

If the return value is IM_NEW_DISPLAY, draws the new current record in the control by sending an LG_DRAW message to self. If the control is one of a pair and wldselwn.flags contains PR_WLDSELWN_RESTRICTED, draws the other control by sending an LG_DRAW message to wldselwn.other. Returns WN_KEY_CHANGED.

WN_SENSE — Sense data

```
VOID wn_sense(SENSE_WLDSELWN *psense);
```

Sense the handle of the database channel and the data for the current record.

The SENSE_WLDSELWN struct is defined as follows:

```
typedef union
    {
    WR_CITY_DATA ci;
    WR_COUNTRY_DATA co;
    } WR_EXTRA_DATA;

typedef struct
    {
    VOID *wfcb;          io channel
    WR_EXTRA_DATA data;
    } SENSE_WLDSELWN;
```

The WR_CITY_DATA and WR_COUNTRY_DATA structs are described under the WR_GET_CITY_DATA and WR_GET_COUNTRY_DATA services respectively in the *Series 3 World Database* chapter of the *I/O Devices Reference* manual.

The significance of the members of the SENSE_WLDSELWN struct is as follows:

wfcb — the handle of the channel to the World database device driver.

data — contains the data for the current record which may be either a city or a country as appropriate.

The method writes the wmatcher.wfcb property of the WMATCHER component to psense->wfcb.

If wldselwn.flags contains IN_WLDSELWN_COUNTRY, writes the data for the current country to the WR_EXTRA_DATA union pointed to by the data member of psense.

Otherwise, if wldselwn.flags does not contain IN_WLDSELWN_COUNTRY, writes the data for the current city to the WR_EXTRA_DATA union pointed to by the data member of psense.

WN_EMPHASISE — Emphasise

```
VOID wn_emphasise(UINT flag);
```

Emphasise the control if flag is TRUE, otherwise de-emphasise it.

If flag is TRUE, sets PR_WIN_EMPHASISED in win.flags. Resets the match string by sending an IM_KEY message to wldselwn.mat, with arguments of W_KEY_ESCAPE and zero. Draws the control by sending an LG_DRAW message to self.

Otherwise, if flag is FALSE, clears the cursor from the control, clears PR_WIN_EMPHASISED from win.flags, and sends an LG_DRAW message to self.

LG_SENSE_WIDTH — Returns required width

```
UINT lg_sense_width(VOID);
```

Return the width of the control.

If the control is a component in a dialog, returns GEOG_MAX_NAME*SYSTEM_FONT_NUM_WIDTH.

Otherwise, returns (GEOG_MAX_NAME-2)*SYSTEM_FONT_NUM_WIDTH.

WLD_RESTRICT — Set restriction

```
VOID wld_restrict(INT IsRestricted);
```

Set restriction.

Writes IsRestricted to the wmatcher.resco property of the WMATCHER component.

If IsRestricted is TRUE, sets PR_WLDSELWN_RESTRICTED in wldselwn.flags, resets the WMATCHER component by writing the NULL string to its matcher.ptyped property, and zero to its matcher.plen property . Draws the control by sending an LG_DRAW message to self.

Otherwise, if IsRestricted is FALSE, clears PR_WLDSELWN_RESTRICTED from wldselwn.flags.

This message may be sent to either partner in a coupled pair: the effect is the same in each case.

This message may not be sent to an isolated control whose wldselwn.flags property contains IN_WLDSELWN_COUNTRIES.

CHAPTER 21

LINK PASTE SUPPORT

This chapter documents the EWLINKSV class which supports extraction of text from the selected region of an edit window, providing a supply of data for the Link Paste mechanism.

Precursors

Familiarity with the following topic will aid the understanding of this chapter:

- the EDWIN class described in the *EDWIN Edit Window Class* chapter of the *HWIM Reference* manual.

EWLINKSV

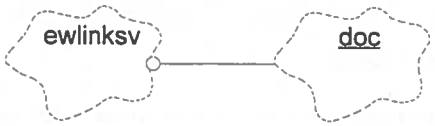
EWLINKSV
doc state pos parend posend buf
ewls_init ewls_extract

The EWLINKSV class supports the extraction of text from the selected region of an edit window as follows:

- the extraction of text including paragraph delimiters - the content of the selected region may thus be copied as one block.
- the extraction of text up to and excluding the next paragraph delimiter - the content of the selected region may thus be copied and stripped of paragraph delimiters - and optionally the replacement of tab characters in the text with spaces.

The class may be used when implementing an edit window as a link-paste server.

Class diagram



Class definition

Defined in sub-category file *edwin.cl* (generated header file *edwin.g*).

```
CLASS ewlinksv root
Component of linksv class, extracts data from edwin
    {
    ADD ewls_init
    ADD ewls_extract
    PROPERTY
        {
        VOID *doc;
        WORD state;
        UWORD pos;
        UWORD parend;
        UWORD posend;
        TEXT buf[256];
        }
    }
```

Property

`ewlinksv.doc`	the handle of a document - this is usually an instance of either `EPDOC` or `EPFDOC`. The document is assumed to be a component in an edit window - i.e. an instance of `EDWIN`.
`ewlinksv.state`	see the description of the `EWLS_EXTRACT` message for the significance of `ewlinksv.state`.
`ewlinksv.pos`	the current position in the document.
`ewlinksv.parend`	for use by subclassers.
`ewlinksv.posend`	the end position of the selected region.
`ewlinksv.buf`	a buffer that stores text extracted from the document.

EWLINKSV methods

EWLS_INIT — Initialise

```
VOID ewls_init(PR_EDWIN *edwin,INT state);
```

Determine the position and length of the selected region in the document edited by the edit window with handle `edwin` and record the extraction mode specified by `state`.

Records the handle of the document by writing `edwin->edwin.doc` to `ewlinksv.doc`.

Writes the start position of the selected region to `ewlinksv.pos` and writes the end position of the selected region plus one to `ewlinksv.posend` - the method obtains the start and end positions of the selected region by sending an `SI_GET_SELECT` message to `edwin->edwin.scrimg`.

Records the extraction mode by writing `state` to `ewlinksv.state`.

EWLS_EXTRACT — Extract some data

```
INT ewls_extract(TEXT **ppb,UINT blen);
```

Extract up to `blen` characters from position `ewlinksv.pos` in the selected region of the document and write the address of the extracted characters to *ppb.

If no region is selected, returns -1.

If the maximum number of characters that can be extracted from the document - `ewlinksv.posend-ewlinksv.pos` - is less than `blen`, writes `ewlinksv.posend-ewlinksv.pos` to `blen`.

Extracts `blen` characters from position `ewlinksv.pos` in the document by sending an `EP_EXTRACT` message to `ewlinksv.doc`. The characters are written to `ewlinksv.buf`.

Writes `&ewlinksv.buf[0]` to `*ppb`.

If `ewlinksv.state` is equal to `DF_EQUAL_PARAS`, increments `ewlinksv.pos` by the number of characters extracted from the document and returns the number of characters extracted.

If `ewlinksv.state` is equal to `DF_LINK_TEXT`, replaces any tab characters in `ewlinksv.buf` with spaces.

If the text written to `ewlinksv.buf` contains no paragraph delimiters - i.e. `0x00` - increments `ewlinksv.pos` by `blen` and returns `blen`.

Otherwise increments `ewlinksv.pos` by one plus the number of characters *before* the first paragraph delimiter in `ewlinksv.buf` and returns the number of characters *before* the first paragraph delimiter.

CHAPTER 22

REPRESENTATIONS OF TIME AND DATE

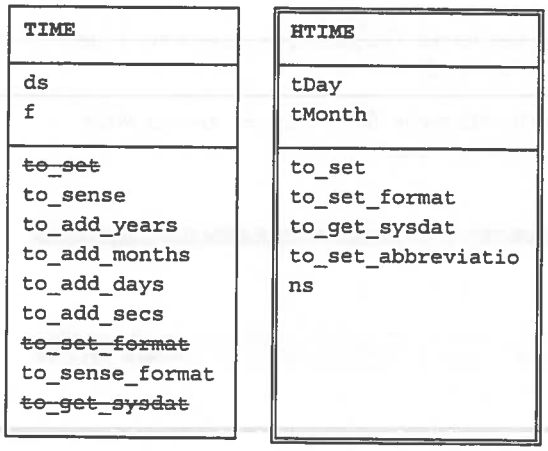
This chapter documentsthe HTIME class which may be used to convert to and from textual representations of time and date.

Precursors

Familiarity with the following topic will aid the understanding of this chapter:

- the TIME class described in the *The Time Class* chapter of the *OLIB Reference* manual.

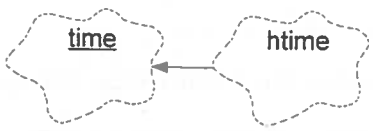
HTIME



The HTIME class supports conversion to and and from textual representations of time where *time* is used to denote time and date. While based on the TIME class in the OLIB object library the HTIME class adds the following functionality:

- the ability to truncate the day and month names after a given number of characters.
- the ability to sense machine specific format parameters such as time and date separator characters.

Class diagram



Class definition

Defined in sub-category file *htime.cl* (generated header file *htime.g*).

```
CLASS htime time
    {
    REPLACE to_set
    REPLACE to_set_format
    REPLACE to_get_sysdat
    ADD to_set_abbreviations

    CONSTANTS
        {
        PR_TIME_ABBREVIATE          0x1000
        PR_TIME_FORCE_UPDATE_FORMAT 0x2000
        HTIME_FIRST_DAY         29219L          1st Jan 1980
        HTIME_LAST_DAY          54786L          31st Dec 2049
        HTIME_FIRST_SEC         0L
        HTIME_LAST_SEC          86399L          (P_NSECDAY-1)
        HTIME_FIRST_MIN         0L
        HTIME_LAST_MIN          86340L          (P_NSECDAY-60)
        }

    PROPERTY
        {
        UBYTE tDay;
        UBYTE tMonth;
        }
    }
```

Property

`htime.tDay` the length of the abbreviated form of the day name. Thus if `htime.tDay` were 3 the abbreviated form of "Wednesday" would be "Wed".

`htime.tMonth` the length of the abbreviated form of the month name. Thus if `htime.tMonth` were 3 the abbreviated form of "December" would be "Dec".

HTIME methods

TO_SET — Set time

```
INT to_set(INT format,UBYTE *pdata);
```

Set the current date and time using the representation specified by `format`.

If `format` contains `SET_TIME_NOW` and `time.f.flags` contains `PR_TIME_FORCE_UPDATE_FORMAT`, sets as many as possible of the format parameters using system services by sending `self` a `TO_SET_FORMAT` message with an argument of `NULL`.

Sets the current date and time stored in the `HTIME` object by supersending a `TO_SET` message with arguments of `format` and `pdate` and returns the return value.

TO_SET_FORMAT — Set time format

```
VOID to_set_format(SE_TIME_FORMAT *pf,UWORD mask);
```

Set the format parameters for the conversion to and from textual representations of time.

Sets the format parameters by supersending a `TO_SET_FORMAT` message with arguments of `pf` and `mask`.

Sets the following in `time.f.flags` if present in `pf->flags`:

`PR_TIME_ABBREVIATE` indicates that the day of the week text is to be truncated after `htime.tDay` characters,

and that the day of the month text is to be truncated after `htime.tMonth` characters.

PR_TIME_FORCE_UPDATE_FORMAT controls the behaviour of the TO_SET message.

TO_GET_SYSDAT — Get system data

```
INT to_get_sysdat(UBYTE *buf,UINT type,UINT n);
```

Write a zero terminated time-related name to buf or, if type is TY_TIME_FORMAT write machine specific format parameters in an SE_TIME_FORMAT struct to buf.

If type is neither TY_TIME_DAY, nor TY_TIME_MONTH nor TY_TIME_FORMAT, supersends a TO_GET_SYSDAT message with arguments of buf, type and n, and returns the return value.

If type is TY_TIME_SUFFIX obtains the day suffix name - e.g.. "st", "nd", "rd" or "th" on English language machines - corresponding to the day number specified by n in the range 0 to 31. Writes the string to buf and returns the length of the string.

If type is TY_TIME_AMPM obtains the am/pm string specified by n - where 0 is am and 1 is pm. Writes the string to buf and return the length of the string.

If type is TY_TIME_DAY obtain the day text - e.g. "Tuesday" - for the day number specified by n in the range 0 to 6 inclusive where 0 is Monday. If time.f.flags contains PR_TIME_ABBREVIATE truncates the day text after htime.tDay characters. Writes the string to buf and returns the length of the string.

If type is TY_TIME_MONTH, obtains the month text - e.g. "March" - for the month number specified by n in the range 0 to 11 inclusive where 0 is January. If time.f.flags contains PR_TIME_ABBREVIATE truncates the month text after htime.tMonth characters. Writes the string to buf and returns the length of the string.

If type is TY_TIME_FORMAT, creates an SE_TIME_FORMAT struct and copies the content of the struct to buf. The SE_TIME_FORMAT struct is defined as follows:

```
typedef struct
    {
    UWORD flags;
    UBYTE dsep;
    UBYTE tsep;
    } SE_TIME_FORMAT;
```

The members of the SE_TIME_FORMAT struct are set as follows:

flags — on all machines the following flags are set:

PR_TIME_DAY_NAME	write the day name with a trailing comma before the date.
PR_TIME_SUFFIX_NAME	write the day number followed by a suffix e.g. 15th.
PR_TIME_MONTH_NAME	show the month as a name rather than a numer.

the following flags may be set depending on the machine and the current settings:

PR_TIME_MMDDYY	set on USA machines only.
PR_TIME_YYMMDD	set on Japanese machines only.
PR_TIME_DDMMYY	set on all other machines including European.
PR_TIME_AMPM	set if the system time format is am/pm.
PR_TIME_ABBREVIATE	set if present in time.f.flags.
PR_TIME_FORCE_UPDATE_FORMAT	set if present in time.f.flags.

dsep — the system date separator character - on English language machines this is usually a backslash - 12/3/1994 for example.

tsep — the system time separator character - on English language machines this is usually a colon - 3:42 pm for example.

Returns the length of the data written to buf.

TO_SET_ABBREVIATIONS — Set abbreviations

```
VOID to_set_abbreviations(INT aDay,INT aMonth);
```

Set the length of the day and the month strings.

Writes `aDay` to `htime.tDay` and writes `aMonth` to `htime.tMonth`.

CHAPTER 23

THE GATE CLASS

GATE	
waotstat	lastmods
concb	atpid
xadd	atson
flags	today_hook
dialres	shift_txt
x	prevdisp
getkeys	dx
lastkey	g
gt_init	gt_choice_close
gt_menu_open	gt_button_open
gt_menu_add	gt_button_add
gt_menu_run	gt_button_close
gt_menu_close	gt_dial_position
gt_dial_open	gt_xw_enable
gt_dial_add	gt_return_signals
gt_dial_run	gt_check_ats_on
gt_dial_close	gt_last_key
gt_card_open	gt_wdr_print
gt_card_add	gt_wdr_print_setup
gt_card_close	gt_do_help
gt_choice_open	gt_set_rscfile
gt_choice_add	gt_dial_sopen

The GATE class is essentially a private class whose principal task is to support the graphics functionality of OPL/w and the HWIF library. In consequence, the majority of its methods and items of property are not suitable for use by application programmers. This chapter documents only those aspects that could reasonably be of value to such programmers.

Note that none of the features documented in this chapter are available on the Series 3.

All HWIM applications create and initialise an instance of the GATE class during the application's initialisation. This instance remains in existence for the lifetime of the application. The handle of the application's instance of the GATE class is written to the magic static DatGate.

In the interest of future compatibility, an application should not subclass GATE.

Class definition

Defined in sub-category file *gate.cl* (generated header file *gate.g*).

```
CLASS   gate    root
Gateway to hwim menus and dialogs
    {
    ADD gt_init
    ADD gt_menu_open
    ADD gt_menu_add
    ADD gt_menu_run
    ADD gt_menu_close
    ADD gt_dial_open
    ADD gt_dial_add
    ADD gt_dial_run
    ADD gt_dial_close
    ADD gt_card_open
    ADD gt_card_add
    ADD gt_card_close
    ADD gt_choice_open
    ADD gt_choice_add
    ADD gt_choice_close
    ADD gt_button_open
    ADD gt_button_add
    ADD gt_button_close
    ADD gt_dial_position
    ADD gt_xw_enable
    ADD gt_return_signals
    ADD gt_check_ats_on          check if process supports ATS
    ADD gt_last_key              return last key used
    ADD gt_wdr_print
    ADD gt_wdr_print_setup
    ADD gt_do_help
    ADD gt_set_rscfile
    ADD gt_dial_sopen            Workabout ONLY - init dlg with small font(Roman8)

    CONSTANTS
        {
        PR_XWSERV_SHUTDOWN    0x01
        PR_XWSERV_QUERY_ALERT 0x02
        PR_XWSERV_S3FS        0x04       Workabout ONLY - full-screen S3
compatibility
        PVV_TEXT_MAX_LEN      50
        OPRINTER_DO_PREVIEW     1
        OPRINTER_PRINT_AFTER    2
        }

    TYPES
        {
        typedef struct
            {
            WORD class;
            VOID *data;
            } DITEM_LIST;
        typedef struct
            {
            UWORD xwim;
            UWORD nmenus;
            VOID **menus;
            UWORD ulines[16];
            UWORD count;
            DITEM_LIST dialret[9];
            UWORD nrec;
            VOID *orec;
            UWORD preview;
            } PURE_GATE;
        typedef struct
            {
            UBYTE Mode;
            UBYTE Flags;
            } PVV_DISPLAY;
```

```
        typedef struct
            {
            UWORD fht;      height of system font
            UWORD fds;      descent of system font
            UWORD fas;      ascent of system font
            UWORD fnw;      width of numeric character in system font
            UWORD fmw;      width of maximum normal character
            UWORD cht;      height of a line in dialogs or listboxes
            UWORD mbht;     height of menubar
            UWORD bfid;     id of bold font
            UWORD bulyo;    y-offset to start of bullet
            UWORD bulht;    height of bullet
            UWORD ninlb;    max number of lines in listbox
            UWORD ssww;     width of small status window
            UWORD bsww;     width of large status window
            UWORD hllg;     help list left gutter
            UWORD samo;     menu offset for single accelerator
            UWORD damo;     menu offset for double accelerator
            } X_CONSTANTS;
        typedef struct
            {
            UWORD fht;      height of font
            UWORD fds;      descent of font
            UWORD fas;      ascent of font
            UWORD fnw;      width of numeric character
            UWORD fmw;      width of maximum normal character
            UWORD cht;      height of a line in dialogs or listboxes
            UWORD bulyo;    y-offset to start of bullet
            UWORD bulht;    height of bullet
            UWORD ninlb;    max number of lines in listbox
            } DLG_X_CONSTANTS; Workabout ONLY - extras for dialogs using another font
        }

    PROPERTY
        {
        VOID *waotstat;
        VOID *concb;
        HANDLE xadd;
        UWORD flags;
        UWORD dialres;
        X_CONSTANTS x;          connection pseudo-constants
        VOID *getkeys;          where to send copy of key received
        UWORD lastkey;          last key received
        UWORD lastmods;         last keyboard modifiers received
        UWORD atpid;            pid of process attached to
        UWORD atson;            ATS supported
        VOID *today_hook;
        TEXT shift_txt[10];
        PVV_DISPLAY prevdisp;
        DLG_X_CONSTANTS dx;     Workabout ONLY - constants of additional fonts of
dialog
        PURE_GATE g;
        }
    }
```

Property

`gate.x` — Pseudo-constant connection data, described below. This data may be read by application code.

`gate.getkeys` — Either NULL or the handle of the instance of the XADD ATSSV class that is sent a copy of each key received.

`gate.lastkey` — The keycode of the last key received. This data should be read by means of the `gt_last_key` method.

`gate.lastmods` — The modifiers associated with the last key received. This data should be read by means of the `gt_last_key` method.

`gate.atpid` — Either zero or the process ID of the process to which the application is attached.

`gate.atson`	TRUE if the application supports the automatic test system (ATS) mechanism.
`gate.dx`	Pseudo-constant connection data, described below, for Work*about* dialogs. This data may be read by Work*about* application code.

The remaining items of property are reserved for internal use and are not documented and should not be accessed by application software.

The pseudo-constant connection data, `gate.x`, is a collection of values that are constant within a particular running application, but will, in general, vary with the type of the machine on which the application is running.

The data is evaluated during the application's initialisation and is used by system code to determine the position and size of system-created objects, such as dialog boxes and the menu bar. Application code is free to read any of this data.

The meaning of each item is given in the following table:

`DatGate->gate.x.fht`	The height of the system font.
`DatGate->gate.x.fds`	The descent of the system font.
`DatGate->gate.x.fas`	The ascent of the system font.
`DatGate->gate.x.fnw`	The width of a numeric character in the system font.
`DatGate->gate.x.fmw`	The width of of the widest 'normal' character.
`DatGate->gate.x.cht`	The height of a line in a dialog or a listbox.
`DatGate->gate.x.mbht`	The height of the menubar.
`DatGate->gate.x.bfid`	The ID of the bold font.
`DatGate->gate.x.bulyo`	The y-offset to the top of a bullet displayed in a text window (an instance of TEXTWIN).
`DatGate->gate.x.bulht`	The height of a bullet displayed in a text window (an instance of TEXTWIN).
`DatGate->gate.x.ninlb`	The maximum number of lines that can appear in a listbox.
`DatGate->gate.x.ssww`	The width of the small status window.
`DatGate->gate.x.bsww`	The width of the large status window.
`DatGate->gate.x.hllg`	The width of the left gutter of a help list.
`DatGate->gate.x.samo`	The offset, from the right hand side of a pull-down menu, to the position to display an unshifted accelerator.
`DatGate->gate.x.damo`	The offset, from the right hand side of a pull-down menu, to the position to display the 'Shift' text for a shifted accelerator.

The Work*about* pseudo-constant connection data, `gate.dx`, is a collection of values that are used to determine the appearance of system-crated objects, particularly dialogs, that use a font other than the system font. These values are constant within a particular running application. Although currently only defined for the Work*about*, they will, in general, vary with the type of the machine on which the application is running.

The data is evaluated during the application's initialisation. Application code is free to read any of this data.

The meaning of each item is given in the following table:

`DatGate->gate.dx.fht`	The height of the dialog font.
`DatGate->gate.dx.fds`	The descent of the dialog font.
`DatGate->gate.dx.fas`	The ascent of the dialog font.
`DatGate->gate.dx.fnw`	The width of a numeric character in the dialog font.
`DatGate->gate.dx.fmw`	The width of of the widest 'normal' character in the dialog font.

`DatGate->gate.dx.cht`	The height of a line in a dialog or a listbox.
`DatGate->gate.dx.bulyo`	The y-offset to the top of a bullet displayed in a text window (an instance of `TEXTWIN`).
`DatGate->gate.dx.bulht`	The height of a bullet displayed in a text window (an instance of `TEXTWIN`).
`DatGate->gate.dx.ninlb`	The maximum number of lines that can appear in a listbox.

GATE methods

GT_CHECK_ATS_ON — Check ATS status

```
INT gt_check_ats_on(INT pid);
```

Return the ATS status of the process with process ID `pid`.

Returns zero if the specified process supports ATS, otherwise a (negative) error number. A return value of `E_GEN_NSUP` indicates that the specified process does not support ATS. Other errors are possible if, for example, the specified process has terminated before the `gt_check_ats_on` method is executed.

Note that the process with process ID `pid` may terminate at any time. A zero return value from the `gt_check_ats_on` is thus no guarantee that future ATS operations on that process will succeed.

GT_LAST_KEY — Sense the last keypress

```
INT gt_last_key(UWORD *mods);
```

Sense the keycode and modifiers of the last keypress received by the application.

The method returns the keycode and writes, to `*mods`, the corresponding modifier flags. Both values will be zero if the application has not received any keypress.

CHAPTER 24

HWIM UTILITY FUNCTIONS

HWIM utility and convenience functions have been developed in response to a number of observations and pressures:

- The need to minimise memory used by application code.
- The observation that a large range of applications have many code fragments that are in common use.

By formalising these code fragments into commonly available functions, application code can be made easier to read and can achieve savings in memory.

At first sight, the use of "free standing" functions seems to violate the most basic principles of object oriented programming. This is not the case. They can legitimately be used where similar actions are done repeatedly in various parts of the application code, even if that action involves the sending of messages.

For example, utility functions are very often used to send a standard message to system generated objects such as the application manager which exist throughout the lifetime of an application.

There is nothing to stop a developer from writing his/her own utility functions and is, in fact, encouraged to do so. It is appropriate in a situation where similar pieces of code are used over and over again. It is particularly suitable where code involving the sending of a message to a particular object may be repeated. There is no problem involved in encapsulating the sending of messages within utility functions.

The use of utility functions can save memory by:

- allowing a reduction in the number of parameters passed from application code. While this might save one or two bytes per call, the total saving in memory across a whole range of applications can be substantial
- avoiding the duplication of code fragments

The next sections describe the utility functions available for use by any HWIM application. Each section describes a set of functions grouped under the following headings:

- General utilities
- Text management
- User notification
- Running a dialog
- Dialog box utilities

In many cases, the implementation of utility functions is given in full or in skeleton form. This will assist developers to create their own utility/convenience functions.

Note that the HWIM utility functions are prototyped in *hwim.h*.

General utilities

As indicated by the title of this section, this is a group of miscellaneous functions with no connecting theme.

p_true — Return TRUE

```
INT p_true(VOID);
```

This is a simple function that returns the value TRUE.

It is particularly useful as a default method for a class where the method must return a simple TRUE or FALSE value.

For example, in the definition of the HWIM comman class, p_true is assigned to the method com_accl_check which means that when the method com_accl_check is called, p_true is executed, thus returning a value of TRUE.

p_false — Return FALSE

```
INT p_false(VOID);
```

This is a simple function that returns the value FALSE.

It is particularly useful as a default method for a class where the method must return a simple TRUE or FALSE value.

See the earlier description of p_true for an example of how this could be implemented.

hDestroy — Destroy an object

```
VOID hDestroy(VOID *hand);
```

This is a function which will destroy an object by sending it the DESTROY message:

```
psend2(hand,O_DESTROY);
```

The handle of the object to be destroyed must be passed as a parameter to this function. If the handle is NULL, the function does nothing.

hInitVis — Make a window visible

```
VOID hInitVis(VOID *win);
```

This can be used in one of two ways depending on the current state of the window object with handle win:

- if the window has not yet been initialised, then it will be both initialised and made visible
- if the window has already been initialised, then it will be made visible.

The function is implemented by sending a WN_VISIBLE message to the window with handle win as shown below. The handle passed as a parameter must point to an object instanced from the win class or a subclass of win.

```
psend3(win,O_WIN_VISIBLE,WV_INITVS);
```

For further discussion on windows, see the *Windows* chapter.

hWservComSend — Send command to command manager

```
VOID hWservComSend(INT comid);
```

This utility simply sends the message with message (method) number comid and the *same* message (method) number as a *parameter* to the command manager.This is implemented as shown below:

```
p_send3(w_ws->wserv.com,comid,comid);
```

The additional `comid` parameter that is passed with the message is for the convenience of the receiving command manager method, which may choose to ignore it.

This function is particularly useful where a number of methods are implemented using the same code. Passing the method number as a parameter allows the code to identify the context of the call and to take appropriate action if it so wishes.

For further discussion on the command manager, see the *Command Manager* chapter of this manual and the *Commands and Command Menus* chapter of the *Object Oriented Programming Guide*.

hEnsurePath — Ensure path exists

```
VOID hEnsurePath(TEXT *fname);
```

If the path indicated by the file specification at `fname` does not exist, the function attempts to create the required directory structure.

The file specification is parsed using the Plib function `p_fparse`. If this is successful, the directory component of the parsed file specification is created, provided it does not already exist, using the Plib function `p_mkdir`.

No errors are reported. If this function fails, this could be due to a bad file specification in `*fname` or a problem with the device/medium when attempting to create the directory itself.

For example, suppose `fname` contains the string:

```
"FILES\\DOCS\\FRED.DOC"
```

and the default path is:

```
LOC::M:\
```

then the function will attempt to create the directory structure:

```
\FILES\DOCS\
```

at node LOC:: on device M:

For further information on file specifications see the *files* chapter in the *PLIB Reference* manual.

Text management

This group of functions is concerned with building text from various sources, displaying text in a variety of contexts and with text manipulation.

hLoadResource — Allocate cell and load resource

```
INT hLoadResource(INT rid, VOID *ppdata);
```

This is a convenience routine which allocates a cell of suitable length from the heap, loads the resource referenced by the resource id `rid` into the cell and returns the length of the loaded resource. The address of the cell is placed in `*ppdata`.

The function is implemented by sending an `am_load_resource` message to the application manager as shown below:

```
p_send4(w_am,O_AM_LOAD_RESOURCE,rid,ppdata);
```

Remember that the address of the application manager object is found in the magic static variable `w_am`.

If an error occurs, the function calls `p_leave`.

hLoadResBuf — Load resource into buffer

```
INT hLoadResBuf(INT rid, VOID *buf);
```

This is a convenience routine which loads the resource referenced by the resource id `rid` into the buffer pointed to by `buf` and returns the length of the loaded resource.

The function is implemented by sending an `am_load_res_buf` message to the application manager as shown below:

```
p_send4(w_am,O_AM_LOAD_RES_BUF,rid,buf);
```

It is similar to the previously described function `hLoadresource` except that it is the caller's responsibility to provide the buffer and to ensure that it is large enough to contain the loaded resource.

If an error occurs, the function calls `p_leave`.

As an example, consider the following code fragment that could come from any typical window subclass drawing method which, as part of its functionality, loads a resource string and prints it at a given point within the window:

```
.
.
hLoadResBuf(self->resid,&buf[0]);
p_supersend2(self,O_WN_DRAW);
gPrintText(50,50,&buf[0],p_slen(&buf[0]);
.
.
```

The important point to note here is that `buf` must be large enough to hold the loaded resource string.

hLoadChlistResBuf — Load choice list item into buffer

```
TEXT *hLoadChlistResBuf(INT rid, INT choice, TEXT *buf);
```

This is a convenience routine which loads a *single* choice list item from a choice list resource into a buffer.

The choice list resource is identified by the parameter `rid` within a resource file while the parameter `choice` identifies the particular choice list item *within* the resource.

The function returns a pointer to the string's terminating zero.

The function is implemented by sending a `ws_load_chlist_res` message to the window server object as shown below:

```
return((TEXT *)p_send5(w_ws,O_WS_LOAD_CHLIST_RES,rid,choice,buf);
```

It is the caller's responsibility to ensure that the buffer provided is large enough to contain the loaded choice list item.

If an error occurs, the function calls `p_leave`.

hErrs — Generate error text

```
VOID hErrs(TEXT *buf, INT err);
```

This is a convenience routine that writes the error text specified by the error number `err` to the buffer at `buf`.

If `err` is negative it is interpreted as a system error number and the text is obtained by calling the Plib function `p_errs`; the buffer must be at least `E_MAX_ERROR_TEXT_SIZE` bytes.

If `err` is non-negative it is assumed to refer to a resource string which is loaded from a resource file with resource id `err-ERROR_RID_OFFSET` (see the `WIN` class definition). The buffer supplied must be large enough to contain the maximum size string expected.

Because the resource id is given as `err-ERROR_RID_OFFSET`, then this *must* evaluate to a number greater than 1 in order to access resources within the application resource file. Consequently, `err` must be greater than `ERROR_RID_OFFSET + 1`.

The function is implemented as follows:

```
    .
    .
{
if (err<0)
    p_errs(buf,err);
else
    hLoadResBuf(err-ERROR_RID_OFFSET,buf);
}
   .
   .
```

If an error occurs, the function calls `p_leave`.

For further detail on resource files, see the chapter on *Resource Files* in this manual.

hAtob — Generate formatted string

```
INT hAtob(TEXT *buf, INT rid, INT *pargs);
```

This is a useful function which generates a zero terminated string in the buffer pointed to by `buf` and returns its length. The string is generated from a format string and a number of arguments pointed to by `pargs`. The format string is loaded from the resource `rid` in a resource file.

For more detail on the structure of the format string, see the description of the PLIB function `p_atob` in the *PLIB Reference* manual.

There are a number of important points to note:

- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes.
- it is the caller's responsibility to ensure that the buffer pointed to by `buf` is large enough to contain the generated string.

If an error occurs, the function calls `p_leave`.

The following routine and code fragment together provide a simple example of the use of this function. For the sake of argument, assume that `file` is a static variable that contains the channel of an opened text file.

```
GLDEF_C CDECL SampleRoutine(INT rid,INT arg,...)
    {
    INT  len;
    TEXT buf[25];
    len = hAtob(&buf[0],rid,&arg);
    p_write(file,&buf[0],len);
    }


    .
    .
INT  rid;
INT  num;
    .
    .
num = 65;
rid = MY_FORMAT_STRING;
SampleRoutine(rid,num,num,num,num,num,num);
    .
    .
```

Given that the string resource referenced by the resource ID `MY_FORMAT_STRING` is defined as:

```
RESOURCE STRING my_format_string {str="[%b %c %d %o %u %x]";}
```

then the the following string of characters would be written to the text file:

```
[1000001 A 65 101 65 41]
```

hAtos Generate formatted string, variable argument count

```
TEXT *hAtos(TEXT *buf, INT rid, ...);
```

This is a useful function which generates a zero terminated string in the buffer pointed to by `buf` and returns a pointer to its zero terminator.The string is generated from a format string and a number of arguments. The format string is loaded from the resource `rid` in a resource file while the arguments are interpreted as for the PLIB function `p_atos`.

For more detail on the structure of the format string, see the description of the PLIB functions `p_atob` and `p_atos` in the *PLIB Reference* manual.

As with the function `hAtob`, described earlier, the same important points must be noted. To recap:

- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes.
- it is the caller's responsibility to ensure that the buffer pointed to by `buf` is large enough to contain the generated string.

If an error occurs, the function calls `p_leave`.

For example, given that the string resource referenced by the resource ID `MY_FORMAT_STRING` is defined as:

```
RESOURCE STRING my_format_string {str="%b %c %d %o %u %x";}
```

the effect of the following code fragment:

```
INT  num;
TEXT buf[23];
INT  rid;
    .
    .
num = 65;
rid = MY_FORMAT_STRING;
hAtos(&buf[0],rid,num,num,num,num,num,num);
```

is to build the string:

```
1000001 A 65 101 65 41
```

in the buffer at `buf`.

hAppendEllipsis Append ellipsis to text

```
VOID hAppendEllipsis(TEXT *p);
```

This is a simple convenience function that writes the ellipsis character, `WS_SYMBOL_ELLIPSIS`, followed by a zero terminator into the two bytes pointed to by `p`.

The ellipsis character is commonly used as an appendage to a choice list prompt to indicate that a subsidiary dialog is available.

For example the code fragment:

```
TEXT  buf[5] = {'A','B','C'};
hAppendEllipsis(&buf[3]);
```

would result in `buf` containing the four characters ABC... followed by the zero terminator.

hSetGTmode Set text mode

```
VOID hSetGTmode(UINT tmode);
```

This simple convenience function uses the window server function `gSetGC` to set the text mode of the *current* graphics context to that specified by `tmode`. Possible values of `tmode` are defined in the *Graphics Output* chapter of the *Window Server Reference* manual.

For example, as part of the drawing method of a `win` subclass, the following code fragment would create a a permanent graphics context with default values and then change the textmode so that when writing text, the 1's and 0's of the font characters overwrite the destination area:

```
    .
    .
wValidateWin(self->win.id);
p_supersend2(self,O_WN_DRAW);
self->mywin.gcid = gCreateGC0(self->win.id);
hSetGTmode(G_TRMODE_REPL);
gPrintText(50,50,&buf[0],p_slen(&buf[0]));
wFree(self->mywin.gcid);
    .
    .
```

hSetGFont — Set font

```
VOID hSetGFont(UINT fid);
```

This simple convenience function uses the window server function `gSetGC` to set the text font of the *current* graphics context to that specified by `fid`. Possible values of `fid` are defined in the *Graphics Output* chapter of the *Window Server Reference* manual.

For example, as part of the drawing method of a `win` subclass, the following code fragment would create a a permanent graphics context with default values and then change the text font to the ROM based monospaced font `WS_FONT_BASE+3`:

```
    .
    .
wValidateWin(self->win.id);
p_supersend2(self,O_WN_DRAW);
self->mywin.gcid = gCreateGC0(self->win.id);
hSetGFont(WS_FONT_BASE+3);
gPrintText(50,50,&buf[0],p_slen(&buf[0]));
wFree(self->mywin.gcid);
    .
    .
```

hSetGStyle — Set text style

```
VOID hSetGStyle(UINT style);
```

This simple convenience function uses the window server function `gSetGC` to set the text style of the *current* graphics context to that specified by `style`. Possible values of `style` are defined in the *Graphics Output* chapter of the *Window Server Reference* manual.

For example, as part of the drawing method of a `win` subclass, the following code fragment would create a a permanent graphics context with default values and then change the text style so that text characters are displayed in bold:

```
    .
    .
wValidateWin(self->win.id);
p_supersend2(self,O_WN_DRAW);
self->mywin.gcid = gCreateGC0(self->win.id);
hSetGStyle(G_STY_BOLD);
gPrintText(50,50,&buf[0],p_slen(&buf[0]));
wFree(self->mywin.gcid);
    .
    .
```

hGetBWid — Get normal text width for buffer

```
INT hGetBWid(TEXT *buf, INT len);
```

This function uses the window server function `gTextWidth` to calculate and return the width, in pixels, of the first `len` characters in the buffer pointed to by `buf`. This function assumes that the text would be displayed using the current system font (i.e `SYSTEM_FONT_ID`) and a normal style (i.e `G_STY_NORMAL`).

This is useful for planning the size and positioning of text in relation to surrounding text and graphic objects.

hGetSWid — Get normal text width for string

```
INT hGetSWid(TEXT *pzts);
```

This is similar to the function `hGetBWid`.

It calculates and returns the width, in pixels, of the zero terminated string pointed to by `pzts`. Like `hGetBWid`, this function assumes that the text would be displayed using the current system font (i.e `SYSTEM_FONT_ID`) and a normal style (i.e `G_STY_NORMAL`).

This function is useful for planning the size and positioning of text in relation to surrounding text and graphic objects.

hGetBBWid — Get bold text width for buffer

```
INT hGetBBWid(TEXT *buf, INT len);
```

This is similar to the function `hGetBWid` in that it uses the window server function `gTextWidth` to calculate and return the width, in pixels, of the first `len` characters in the buffer pointed to by `buf`. This function assumes that the text would be displayed using the font with id `BOLD_FONT_ID` and a normal style (i.e `G_STY_NORMAL`).

This is useful for planning the size and positioning of text in relation to surrounding text and graphic objects.

User notification

This group of functions is concerned with informing, warning and alerting the user in a variety of ways.

hInfoPrint — Display an information message

```
VOID hInfoPrint(INT rid, ...);
```

This function prints an information message window in the bottom right hand corner of the screen for 2 to 2.5 seconds or until cancelled.

The zero terminated text of the message is generated from a format string and the arguments which follow the parameter `rid`. The format string is loaded from the resource `rid` in a resource file.

For more detail on the structure of the format string, see the description of the PLIB function `p_atob` in the *PLIB Reference* manual

There are a number of important points to note:

- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes
- if the generated text is greater than `W_INFO_MSG_MAX_LEN` bytes then the message will appear truncated
- the generated text including the zero terminator must not, in any event, be greater than 80 bytes.

If an error occurs, `p_leave` is called.

The following code fragment provides a simple example of the use of this function.

The string in the resource file referenced by the resource id `RID_OF_MY_FORMAT_STRING` contains the 25 character sequence:

```
The numbers are %d and %d
```

the effect of the following code fragment:

```
INT  num1,num2;
INT  rid;
   .
   .
num1 = 65;
num2 = 66;
rid = RID_OF_MY_FORMAT_STRING;
hInfoPrint(rid,num1,num2);
```

is to display the message:

```
The numbers are 65 and 66
```

in the bottom right hand corner of the screen.

hInfoPrintErr — Display an error information message

```
VOID hInfoPrintErr(INT err);
```

This function prints an error information message window in the bottom right hand corner of the screen for 2 to 2.5 seconds or until cancelled.

The text of the message is related to the error number in `err` and is obtained in exactly the same way as described in the function `hErrs`; see the description of `hErrs` for more information.

If an error occurs, `p_leave` is called.

hBusyPrint — Display a busy message

```
VOID hBusyPrint(INT delay, INT rid, ...);
```

This function prints a flashing message window in the bottom left hand corner of the screen. The display of the message can be delayed by the number of *half-seconds* specified in the `delay` parameter; this can range from 0 to 63.

The zero terminated text of the message is generated from a format string and the arguments which follow the parameter `rid`. The format string is loaded from the resource `rid` in a resource file.

The message may be removed by a call to the window server function `wCancelBusyMsg`.

For more detail on the structure of the format string, see the description of the PLIB function `p_atob` in the *PLIB Reference* manual

There are a number of points to note:

- the generated text including the zero terminator must *not* be greater than 30 bytes. (Note that this is less than the 80 bytes common for other functions)
- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes

If an error occurs, `p_leave` is called.

A common use of a flashing message area in the bottom left hand corner of the screen is to display a "busy" type messge to inform the user that the work in progress could take some time to complete.

hBeep — Make a beep

```
VOID hBeep(VOID);
```

This is a convenience function that makes a short beep sound suitable for accompanying an error notification. It can, of course, be used wherever an application sees fit.

The function is implemented by making the following call:

```
p_sound(-5,320);
```

The beep is sounded for a duration of 5 system ticks (that is 5/32 sec) at a frequency of 512/320 KHz (that is 1.6 KHz).

Run a dialog

This group of functions is concerned with starting some standard dialogs and launching dialogs in general.

hLaunchDial — Launch a dialog

```
INT hLaunchDial(P_CATID cat, INT class, DL_DATA *data);
```

This is a convenience routine which loads, initialises and runs a dialog. The parameter `cat` is the category *number* of the dialog class while the parameter `class` is its class number.

The `DL_DATA` struct is defined in *hwimman.g* as:

```
typedef struct
    {
    UWORD id;           /* resource id of a DIALOG resource*/
    VOID *rbuf;         /* address of result buffer, or NULL */
    PR_DLGBOX **pdlg;   /* address of where to write handle of dialog, or NULL */
    } DL_DATA;
```

The function is implemented as shown below by sending a `O_WS_DO_DIAL` message to the `WSERV` object.

```
p_send5(w_ws,O_WS_DO_DIAL,p_getlibh(cat),class,data);
```

`hLaunchDial` returns whatever value that the method `O_WS_DO_DIAL` returns. This will be zero for dialogs cancelled without the intervention of application code.

hConfirm — Run a confirm dialog

```
INT hConfirm(INT rid,...);
```

This is a convenience routine which presents a standard one line query dialog which returns a `TRUE` or `FALSE` value. This type of dialog is often used to ask a user to confirm an intended action.

The text of the dialog is generated from a format string and (optionally) a number of arguments. The format string is loaded from the resource `rid` in a resource file while the arguments (if any) are interpreted as for the PLIB function `p_atos`. More information on `p_atos` can be found in the *PLIB Reference* manual.

The function is implemented as shown below by sending a `O_WS_QUERY_DIALOG` message to the `WSERV` object.

```
p_send5(w_ws,O_WS_QUERY_DIALOG,0,rid,&rid+1);
```

`hConfirm` returns whatever value that the method `O_WS_QUERY_DIALOG` returns. This will be `TRUE` if the user confirms the action.

There are a number of points to note:

- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes.
- the resulting text string must not be greater than 100 bytes.

If the resulting text string is sufficiently large to force the width of the resulting dialog box to exceed the width of the window, then a panic will result.

h2LineConfirm — Run a two-line confirm dialog

```
INT h2LineConfirm(INT secondrid, INT rid, ...);
```

This is a convenience routine which presents a two line query dialog which returns a `TRUE` or `FALSE` value. This type of dialog is often used to ask a user to confirm an intended action.

The function is very similar to the function `hConfirm` described earlier. The first line of text of the dialog is generated from a format string and (optionally) a number of arguments. The format string is loaded from the resource `rid` in a resource file while the arguments (if any) are interpreted as for the PLIB function `p_atos`. The second line of text is basic text and is loaded from the resource `secondrid` in a resource file.

The function is implemented as shown below by sending a O_WS_QUERY_DIALOG message to the WSERV object.

```
p_send5(w_ws,O_WS_QUERY_DIALOG,secondrid,rid,&rid+1);
```

h2LineConfirm returns whatever value that the method O_WS_QUERY_DIALOG returns. This will be TRUE if the user confirms the action.

There are a number of points to note:

- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes.
- the resulting text string must not be greater than 100 bytes.
- the second line of text in the resource file must not be greater than 100 bytes.

If either of the resulting text strings is sufficiently large to force the width of the resulting dialog box to exceed the width of the window, then a panic will result.

hErrorDialog — Run an error dialog

```
INT hErrorDialog(INT err, INT rid, ...);
```

This is a convenience routine which presents an error dialog with text derived from the error number and a string resource. It returns a zero if the dialog was presented successfully. It returns a non-zero value if the dialog presentation failed due to an out of memory error, in which case the dialog remains to be cleaned up (use the OLIB convenience function cl_clean_level, described in *The CLEANUP Class* chapter in the *OLIB Reference* manual).

The text derived from the resource file is generated from a format string and (optionally) a number of arguments. The format string is loaded from the resource with ID rid, while the arguments (if any) are interpreted as for the PLIB function p_atos.

The function is implemented as shown below by sending a O_WS_ERROR_DIALOG message to the WSERV object.

```
p_send5(w_ws,O_WS_ERROR_DIALOG,err,rid,&rid+1);
```

There are a number of points to note:

- the length of the format string in the resource file, when loaded, must *not* be greater than 80 bytes.
- the resulting formatted text string must not be greater than 80 bytes.

This type of dialog is often used for the reporting of errors of a moderately serious nature; minor errors should use hInfoPrintError described earlier and more serious errors should use the application manager's am_notify method.

Dialog box utilities

The following utility functions are available for performing standard operations on the component controls of a dialog. They may be used directly by an application, or used as models for the construction of application-specific utilities.

The code of each used utility function is included in the application. These functions have optimised calling conventions and therefore offer a modest saving over the equivalent function supplied by the application itself.

Note that these utilities assume that the dialog handle is stored in the magic static DatDialogPtr. Thus they may not be used with any dialog which contains the DLGBOX_NO_DDP flag in dlgbox.flags.

hDlgSet — Set an item

```
VOID hDlgSet(UBYTE index, VOID *pset);
```

This function is a generalised way of setting data (or property) into the control of one of the components of the current dialog and is used in the more specialised dialog box utility functions described later.

As stated in the introduction to this section, it is assumed that DatDialogPtr points to the current dialog object.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0). The parameter pset is assumed to point to the appropriate structure containing the information to be set . The class which defines the referenced dialog box component *control*, will have defined a O_WN_SET method and will "know" how to interpret the data pointed to by pset.

The function is implemented by sending a O_WN_SET message to the *current dialog object* as shown below:

```
p_send4(DatDialogPtr,O_WN_SET,index,pset);
```

hDlgSetText — Set a text item

```
VOID hDlgSetText(UBYTE index, TEXT *buf);
```

This function sets text into the control of one of the current dialog's components. The control is assumed to be an instance of TEXTWIN or a subclass.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0). The parameter buf is assumed to point to a zero terminated string containing the text to be set.

This function is implemented as shown below. Note that it uses the more generalised function hDlgSet described earlier.

```
GLDEF_C VOID hDlgSetText(UBYTE index, TEXT *buf)
    {
    SE_TEXTWIN settxt;

    settxt.buf=buf;
    settxt.len=p_slen(settxt.buf);
    settxt.flags=SE_TEXTWIN_TEXT;
    hDlgSet(index,&settxt);
    }
```

hDlgSetEdwin — Set text in edit window

```
VOID hDlgSetEdwin(UBYTE index, Text *buf);
```

This function sets text into the control of one of the current dialog's components. The control is assumed to be an instance of EDWIN or a subclass.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0). The parameter buf is assumed to point to a zero terminated string containing the text to be set.

This function is implemented as shown below. Note that it uses the more generalised function hDlgSet described earlier.

```
GLDEF_C VOID hDlgSetEdwin(UBYTE index, TEXT *buf)
    {
    SE_EDWIN set;

    set.buf=buf;
    set.len=p_slen(set.buf);
    hDlgSet(index,&set);
    }
```

hDlgSetPrompt — Set text in prompt window

```
VOID hDlgSetPrompt(UBYTE index Text *buf);
```

This function sets text into the prompt of one of the current dialog's components.

Note that the component's prompt must exist prior to a call to hDlgSetPrompt. This function merely replaces the text of the prompt and can not be used to add a prompt to a component, if one was not originally specified in the dialog's resource.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0). The parameter `buf` is assumed to point to a zero terminated string containing the prompt text to be set.

This function is implemented as shown below. It may be of interest to point out that the `O_DL_SET_PROMPT` method uses the more generalised function `hDlgSet` described earlier.

```
GLDEF_C VOID hDlgSetPrompt(UBYTE index, TEXT *buf)
    {
    SE_TEXTWIN settxt;

    settxt.buf=buf;
    settxt.len=p_slen(settxt.buf);
    settxt.flags=SE_TEXTWIN_TEXT;
    p_send4(DatDialogPtr,O_DL_SET_PROMPT,index,&settxt);
    }
```

hDlgSetChlist — Set choice in choice list

```
VOID hDlgSetChlist(UBYTE index, INT nsel);
```

This function sets the choice number in a choice list control within one of the current dialog's components, ultimately causing that item in the list to be displayed (or highlighted if the whole list is shown). The parameter `nsel` holds the number of the choice to be set (where a value of 0 refers to the first item in the choice list).

The particular component within the current dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0) and is assumed to be an instance of (or a subclass of) `CHLIST`.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSet` described earlier.

```
GLDEF_C VOID hDlgSetChlist(UBYTE index, INT nsel)
    {
    SE_CHLIST set;

    set.nsel=nsel;
    set.set_flags=SE_CHLIST_NSEL;
    hDlgSet(index,&set);
    }
```

hDlgSetChlistOn — Set On/Off choice list to On

```
VOID hDlgSetChlistOn(UBYTE index);
```

This function is a specialised version of the function `hDlgSetChlist` described earlier.

It sets the choice number in a choice list control to 1.

The particular component within the current dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0) and is assumed to be an instance of `CHLIST` or a subclass.

This function can be used with any choice list but is especially useful if used in conjunction with an Off/On choice list (using the system resource `SYS_OFFON_MENU`). If the current dialog has a component containing this choice list, then using this function (with the appropriate `index`) will set the choice list to On.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSetChlist` described earlier.

```
GLDEF_C VOID hDlgSetChlistOn(UBYTE index)
    {
    hDlgSetChlist(index,1);
    }
```

hDlgSetNcedit — Set value of numeric editor

```
VOID hDlgSetNcedit(UBYTE index, UINT value);
```

This function sets a value into the control of one of the current dialog's components. The control is assumed to be an instance of the numeric editor (NCEDIT) or a subclass.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0).

The value passed to the function in the parameter value can be any valid UINT type.

This function is implemented as shown below. Note that it uses the more generalised function hDlgSet described earlier.

```
GLDEF_C VOID hDlgSetNcedit(UBYTE index, UINT value)
    {
    SE_NCEDIT set;

    set.value=value;
    set.flags=SE_NCEDIT_VALUE;
    hDlgSet(index,&set);
    }
```

hDlgSetLledit — Set latitude/longitude editor

```
VOID hDlgSetLledit(UBYTE index, INT value);
```

This function sets a value into the control of one of the current dialog's components. The control is assumed to be an instance of the latitude/longitude editor (LLEDIT) or a subclass. Whether the dialog control represents a latitude or a longitude depends on the way the editor is initialised.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0).

The value passed to the function should represent the number of *minutes* of latitude or longitude. A positive value represents the number of minutes *North* or *West* while a negative value represents the number of minutes *South* or *East*, respectively.

For a latitude, the magnitude of the value should be no greater than 5399 minutes (that is, 89 degrees and 59 minutes).For a longitude, the magnitude of the value should be no greater than 10799 minutes (that is, 179 degrees and 59 minutes). Larger values may be passsed but the editor will assume the appropriate maximum value.

This function is implemented as shown below. Note that it uses the more generalised function hDlgSet described earlier.

```
GLDEF_C VOID hDlgSetLledit(UBYTE index, UINT value)
    {
    SE_LLEDIT set;

    set.value=value;
    hDlgSet(index,&set);
    }
```

For example, if the third dialog component in the current dialog box contains a longitude editor control and the fourth component contains a latitude editor control, then the code fragment:

```
    .
    .
hDlgSetLledit(2,5110);
hDlgSetLledit(3,-2010);
    .
    .
```

will set the respective editor values to:

85° 10' West and 33° 30' South.

hDlgSetPtedit — Set punctuation editor

```
VOID hDlgSetPtedit(UBYTE index, UBYTE ch);
```

This function sets a value into the control of one of the current dialog's components. The control is assumed to be an instance of the punctuation editor (PUNCTUED) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function allows the single character, passed in the parameter `ch`, to be set. Although an instance of a PUNCTUED class restricts user-keyed input to a valid punctuation character, the value passed in `ch` can be any valid character acceptable to EDWIN, the immediate superclass of PUNCTUED.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSetEdwin` described earlier.

```
GLDEF_C VOID hDlgSetPtedit(UBYTE index, UBYTE ch)
    {
    INT chx;

    chx=ch;
    hDlgSetEdwin(index,(TEXT *)(&chx));
    }
```

For example, if the *third* dialog component in the current dialog box contains a punctuation editor control, then the following call would set the punctuation character to a comma:

```
    .
    .
hDlgSetPtedit(2,',');
    .
    .
```

hDlgSetFledit — Set floating point editor

```
VOID hDlgSetFledit(UBYTE index, DOUBLE *pvalue);
```

This function sets a value into the control of one of the current dialog's components. The control is assumed to be an instance of the floating point editor (FLEDIT) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function allows the floating point number, pointed to by the parameter `pvalue`, to be set. This must be a *valid* double value otherwise the function does nothing.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSet` described earlier.

```
GLDEF_C VOID hDlgSetFledit(UBYTE index, DOUBLE *pvalue)
    {
    SE_FLTEDIT set;

    p_fld(&set.current,pvalue);
    set.set_flags=SE_FLTEDIT_CURRENT;
    hDlgSet(index,&set);
    }
```

hDlgSetDtedit — Set date editor

```
VOID hDlgSetDtedit(UBYTE index, ULONG value);
```

This function sets a value into the control of one of the current dialog's components. The control is assumed to be an instance of the date editor (DTEDIT) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function allows the value passed in the parameter `value`, to be set. The interpretation of the value depends on the nature of the data displayed by the control, as specified during its initialisation (see the description of `DTEDIT` in the *Numeric Editors* chapter). The possible cases are summarised in the following table.

Date editor	`IN_DTEDIT_DDMMYYYY`	Days elapsed since 01/01/1900. *Note that the control can not display dates before* 01/01/1980.
Time of day editor	`IN_DTEDIT_HHMMSS` `IN_DTEDIT_HHMM`	Seconds elapsed since midnight.

Time duration editor	`IN_DTEDIT_HHMMSS_D`
	`IN_DTEDIT_HHMM_D`
	`IN_DTEDIT_HHMMSS_ND`
	`IN_DTEDIT_HHMM_ND`
		Seconds.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSet` described earlier.

```
GLDEF_C VOID hDlgSetDtedit(UBYTE index, ULONG value)
    {
    SE_DTEDIT set;

    set.value=value;
    set.flags=SE_DTEDIT_VALUE;
    hDlgSet(index,&set);
    }
```

hDlgSetRgedit — Set range editor

```
VOID hDlgSetRgedit(UBYTE index, UINT value1, UINT value2);
```

This function sets values into the control of one of the current dialog's components. The control is assumed to be an instance of the range editor (`RGEDIT`) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function allows the upper and lower limits for the range passed in the parameters `value1` and `value2` respectively, to be set. The range editor is a subclass of the `MFNE` (multiple field numeric editor) class.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSet` described earlier.

```
GLDEF_C VOID hDlgSetRgedit(UBYTE index, UINT value1, UINT value2)
    {
    SE_RGEDIT set;

    set.value[IX_RGEDIT_VALUE_1]=value1;
    set.value[IX_RGEDIT_VALUE_2]=value2;
    set.flags=SE_RGEDIT_VALUE_1|SE_RGEDIT_VALUE_2;
    hDlgSet(index,&set);
    }
```

The symbols `IX_RGEDIT_VALUE_1` and `IX_RGEDIT_VALUE_2` are defined in the include file *rgedit.g*.

hDlgSetTitleByRid — Set dialog title

```
VOID hDlgSetTitleByRid(INT rid);
```

The function sets the title of the current dialog box to the zero terminated text referenced by the resource id `rid` in a resource file.

This function is implemented as shown below. The title of a dialog box is always the first component of that dialog box and, therefore, is always referenced by an index value of zero as demonstrated in the implementation of `hDlgSetTitlebyRid`, shown below.

```
GLDEF_C VOID hDlgSetTitleByRid(INT rid)
    {
    TEXT buf[60];

    hLoadResBuf(rid,&buf[0]);
    hDlgSetText(0,&buf[0]);
    }
```

A few points to note:

- the appropriate text string in the resource file, when loaded, cannot be greater than 60 bytes.
- if the text string is sufficiently large to force the width of the resulting dialog box to exceed the width of the window, then a panic will result.

hDlgSense — Sense an item

```
VOID hDlgSense(UBYTE index, VOID *psense);
```

This function is a generalised way of sensing or retrieving data (or property) from the control of one of the components of the current dialog and is used in the more specialised dialog box utility functions described later.

As stated in the introduction to this section, it is assumed that `DatDialogPtr` points to the current dialog object.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0). The parameter `psense` is assumed to point to the appropriate structure in memory into which the information is to be placed. The class which defines the referenced dialog box component *control* will have defined a `O_WN_SENSE` method and will "know" how to interpret the data structure pointed to by `psense`.

The function is implemented by sending a `O_WN_SENSE` message to the *current dialog object* as shown below:

```
p_send4(DatDialogPtr,O_WN_SENSE,index,psense);
```

hDlgSenseEdwin — Sense edit window

```
TEXT *hDlgSenseEdwin(UBYTE index);
```

This function senses the text from the control of one of the current dialog's components. The control is assumed to be an instance of `EDWIN` or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function returns a pointer to a zero terminated string representing the text in the edit window.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSense` described earlier.

```
GLDEF_C TEXT *hDlgSenseEdwin(UBYTE index)
    {
    SE_EDWIN sense;

    hDlgSense(index,&sense);
    return(sense.buf);
    }
```

For example the following code fragment copies the text sensed from an edit window into another buffer; the example assumes that the edit window is contained in the second component in the current dialog box:

```
...
TEXT buf[100];
...
...
p_scpy(&buf[0],hDlgSenseEdwin(1));
...
```

hDlgSenseChlist — Sense a choice list

```
INT hDlgSenseChlist(UBYTE index);
```

This function senses the choice number of the currently highlighted/selected choice item in a choice list control in a component of the current dialog and returns that value. A value of 0 refers to the first item in the choice list.

The particular component within the current dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0). The control is assumed to be an instance of `CHLIST` or a subclass.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSense` described earlier.

```
GLDEF_C INT hDlgSenseChlist(UBYTE index)
    {
    SE_CHLIST sense;

    hDlgSense(index,&sense);
    return(sense.nsel);
    }
```

For example, the following code fragment sets text in an edit window depending on the current item selected in the choice list; the choice list and the edit window being (for the sake of the example) in the current dialog:

```
    .
    .
#define CHOICE_TEXT_YES 1
#define CHOICE_TEXT_NO  2
    .
    .
TEXT buf[2];
    .
    .
buf[1] = '\0';
if (hDlgSenseChlist(2) == CHOICE_TEXT_YES)
    buf[0] = 'Y';
else
    buf[0] = 'N';
hDlgSetText(3,&buf[0]);
```

hDlgSenseNcedit — Sense a numeric editor

```
UINT hDlgSenseNcedit(UBYTE index);
```

This function senses the value from the control of one of the current dialog's components. The control is assumed to be an instance of the numeric editor (`NCEDIT`) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function returns the value currently held by the editor.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSense` described earlier.

```
GLDEF_C UINT hDlgSenseNcedit(UBYTE index)
    {
    SE_NCEDIT sense;

    hDlgSense(index,&sense);
    return(sense.value);
    }
```

hDlgSenseRgedit — Sense a range editor

```
VOID hDlgSenseRgedit(UBYTE index, UWORD *pvalues);
```

This function senses values from the control of one of the current dialog's components. The control is assumed to be an instance of the range editor (RGEDIT) or a subclass.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0).

The function senses the upper and lower limits of the range and places the two values into the first two words of memory pointed to by pvalues. It is the caller's responsibility to supply the two words of memory. The range editor is a subclass of the MFNE (multiple field numeric editor) class.

This function is implemented as shown below. Note that it uses the more generalised function hDlgSense described earlier.

```
GLDEF_C VOID hDlgSenseRgedit(UBYTE index, UWORD *pvalues)
    {
    SE_RGEDIT sense;

    hDlgSense(index,&sense);
    *pvalues++=sense.value[IX_RGEDIT_VALUE_1];
    *pvalues=sense.value[IX_RGEDIT_VALUE_2];
    }
```

The symbols IX_RGEDIT_VALUE_1 and IX_RGEDIT_VALUE_2 are defined in the include file *rgedit.g*.

hDlgSenseFledit — Sense a floating point editor

```
VOID hDlgSenseFledit(UBYTE index, DOUBLE *pvalue);
```

This function senses the value from the control of one of the current dialog's components. The control is assumed to be an instance of the floating point editor (FLEDIT) or a subclass.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0).

The function senses the floating point value and places it in the memory pointed to by pvalue. It is the caller's responsibility to supply this memory location.

This function is implemented as shown below. Note that it uses the more generalised function hDlgSet described earlier.

```
GLDEF_C VOID hDlgSenseFledit(UBYTE index, DOUBLE *pvalue)
    {
    hDlgSense(index,pvalue);
    }
```

hDlgSenseLledit — Sense a latitude/longitude editor

```
INT hDlgSenseLledit(UBYTE index);
```

This function senses a value from the control of one of the current dialog's components. The control is assumed to be an instance of the latitude/longitude editor (LLEDIT) or a subclass. Whether the dialog component represents a latitude or a longitude depends on the way the editor is initialised.

The particular component within the dialog is identified by the index parameter (the first component is identified by an index value of 0).

The function returns the number of *minutes* of latitude or longitude. A positive value represents the number of minutes *North* or *West* while a negative value represents the number of minutes *South* or *East* respectively.

For example, a returned latitude value of -2010 means 33° 30' South while a returned longitude value of 5110 means 85° 10' West.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSet` described earlier.

```
GLDEF_C INT hDlgSenseLledit(UBYTE index)
    {
    SE_LLEDIT sense;

    hDlgSense(index,&sense);
    return(sense.value);
    }
```

hDlgSensePtedit — Sense a punctuation editor

```
INT hDlgSensePtedit(UBYTE index);
```

This function senses a value from the control of one of the current dialog's components. The control is assumed to be an instance of the punctuation editor (`PUNCTUED`) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function returns two bytes: the character currently contained within the punctuation editor and a terminating zero.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSenseEdwin` described earlier.

```
GLDEF_C UINT hDlgSensePtedit(UBYTE index)
    {
    return(*hDlgSenseEdwin(index));
    }
```

hDlgSenseDtedit — Sense a date editor

```
ULONG hDlgSenseDtedit(UBYTE index);
```

This function senses a value from the control of one of the current dialog's components. The control is assumed to be an instance of the date editor (`DTEDIT`) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function returns the current value of the date editor. The interpretation of the return value depends on the nature of the data displayed by the control, as specified during its initialisation (see the description of `DTEDIT` in the *Numeric Editors* chapter). The possible cases are summarised in the following table.

Date editor	`IN_DTEDIT_DDMMYYYY`	Days elapsed since 01/01/1900. *Note that the control can not display dates before* 01/01/1980.
Time of day editor	`IN_DTEDIT_HHMMSS` `IN_DTEDIT_HHMM`	Seconds elapsed since midnight.

Time duration editor	`IN_DTEDIT_HHMMSS_D`
	`IN_DTEDIT_HHMM_D`
	`IN_DTEDIT_HHMMSS_ND`
	`IN_DTEDIT_HHMM_ND`
		Seconds.

This function is implemented as shown below. Note that it uses the more generalised function `hDlgSense` described earlier.

```
GLDEF_C ULONG hDlgSenseDtedit(UBYTE index)
    {
    SE_DTEDIT sense;

    hDlgSense(index,&sense);
    return(sense.value);
    }
```

hDlgItemDim — Dim/undim an item

```
VOID hDlgItemDim(UBYTE index, INT flag);
```

This function changes an aspect of one of the current dialog's components. No assumption is made about the type of component.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function sets the *dim* status of the dialog box component if a value of `TRUE` is passed in parameter `flag`, otherwise it sets the *undimmed* status.

The dialog box class `DLGBOX` implements dimming by removing the bullet which normally precedes the component's prompt and blanks out the component's control.

This function is implemented as shown below by sending the current dialog a `O_DL_ITEM_DIM` message.

```
p_send4(DatDialogPtr,O_DL_ITEM_DIM,index,flag);
```

hDlgItemLock — Lock/unlock an item

```
VOID hDlgItemLock(UBYTE index, INT flag);
```

This function changes an aspect of one of the current dialog's components. No assumption is made about the type of component.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function behaves in a similar fashion to `hDlgItemDim` in that it sets the *locked* status of the dialog box component if a value of `TRUE` is passed in parameter `flag`, and sets the *unlocked* status otherwise.

The dialog box class `DLGBOX` implements locking in a similar way to dimming by removing the bullet which normally precedes the component's prompt However, it does not blank out the component's control but does prevent it from being emphasised .

This function is implemented as shown below by sending the current dialog a `O_DL_ITEM_LOCK` message.

```
p_send4(DatDialogPtr,O_DL_ITEM_LOCK,index,flag);
```

hDlgTakeFocus — Change focus

```
VOID hDlgTakeFocus(UBYTE index);
```

This function changes an aspect of one of the current dialog's components. No assumption is made about the type of component.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function changes the focus to the dialog box component identified by the `index` parameter. In general, this causes most keyboard activity to be directed to that component (with the usual exception of key and modifier combinations set to be captured by other processes).

This function is implemented as shown below by sending the current dialog a `DL_TAKE_FOCUS` message.

```
p_send4(DatDialogPtr,O_DL_TAKE_FOCUS,index);
```

This function, like the `dl_take_focus` method itself, is not suitable for being called from the `dl_dyn_init` method. If an application wishes to set the focus on initialisation, it should do so from a replaced `dl_set_size` method. It may be called from any other method (such as `dl_key`) once the dialog has been made visible.

hDlgSetTwips Set floating point editor from twips value

```
VOID hDlgSetTwips(UBYTE index, UINT value);
```

This function sets a value into the control of one of the current dialog's components. The control is assumed to be an instance of the floating point editor (`FLEDIT`) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The number in the parameter `value` is assumed to be in twip units (that is, 1/1440 inch or 1/567 cm and generally relevant to word processing type applications). The function converts this value into a floating point measurement in the current measurement units, either inches or centimetres. The resulting floating point number is set in the floating point editor.

hDlgSenseTwips Sense twips value from floating point editor

```
UINT hDlgSenseTwips(UBYTE index);
```

This function senses the value from the control of one of the current dialog's components. The control is assumed to be an instance of the floating point editor (`FLEDIT`) or a subclass.

The particular component within the dialog is identified by the `index` parameter (the first component is identified by an `index` value of 0).

The function senses the floating point value. This is assumed to contain a measurement in the current measurement units, either inches or centimetres. This floating point value is converted into twip units (that is, 1/1440 inch or 1/567 cm) and is returned by the function.

INDEX

AM_CLEAN_UP, 2-6
AM_ENSURE_IPCS, 2-8
AM_FINDIMG, 2-7
AM_INIT, 2-3
AM_NEW_FILENAME, 2-7
AM_NOTIFYERR, 2-7
AM_RSCNAME, 2-5
AM_WAIT, 2-6
AM_YIELD, 2-7
AO_ABRUN, 3-30
AO_CANCEL, 3-10, 19-6
AO_INIT, 3-9, 3-29, 13-3, 19-5
AO_QUEUE, 3-10, 13-4, 17-2, 17-4
AO_RUN, 3-11, 3-30
COM_ACCL_CHECK, 4-3
COM_EXIT, 4-5
COM_FILE_CHANGE, 4-4
COM_INIT, 4-3
COM_MENU, 4-3
COM_MODE_CHANGE, 4-4
COM_STATWIN, 4-3
DESTROY, 5-4, 5-12, 6-6, 6-14, 7-11, 7-24, 8-10, 8-19, 8-26, 10-6, 12-13, 14-6, 16-6, 16-12, 17-7, 18-9, 20-5, 20-8, 20-12
DL_CHANGED, 7-21, 15-12, 15-16, 16-14, 16-35, 16-38
DL_DIMMED_MESSAGE, 7-19
DL_DYN_INIT, 7-21, 7-24, 15-4, 15-6, 15-7, 15-11, 15-14, 15-20, 15-23, 16-13, 16-16, 16-21, 16-24, 16-27, 16-30, 16-34, 16-37, 16-41, 17-16
DL_FOCUS, 7-21
DL_HANDLE_TO_INDEX, 7-19
DL_INDEX_TO_HANDLE, 7-20
DL_INIT, 7-11
DL_INQ_MINSIZE, 7-20, 16-22
DL_ITEM_ADD, 7-12, 7-24, 15-6, 17-15
DL_ITEM_APPEND, 7-13
DL_ITEM_DIM, 7-19
DL_ITEM_LOCK, 7-19
DL_ITEM_NEW, 7-22
DL_ITEM_REPLACE, 7-13
DL_KEY, 7-16, 7-25, 15-8, 15-11, 15-15, 15-20, 15-23, 16-14, 16-17, 16-22, 16-25, 16-27, 16-31, 16-35, 16-38, 17-10, 17-12, 17-16
DL_LAUNCH_SUB, 7-21, 16-12, 16-21, 16-31, 16-35
DL_SET_ITEM_FLAGS, 7-20
DL_SET_PROMPT, 7-14
DL_SET_SIZE, 7-18, 15-16, 16-14, 16-42, 17-6
DL_TAKE_FOCUS, 7-17
EW_BRING_IN, 10-22
EW_EP_INSERT, 10-23
EW_EVALUATE, 10-21
EW_FIND, 10-20
EW_INIT_STYLE, 10-24
EW_INSERT, 10-19
EW_LEAVE, 10-20
EW_PASTE_CLIP, 10-22
EW_READONLY, 10-24
EW_REPLACE, 10-20
EW_REPLACE_CLIP, 10-22
EW_RETURN_KEY, 10-23
EW_SENSE, 10-18
EW_SENSE_SIZE, 10-16
EW_SET, 10-17
EW_SET_FONT, 10-19
EW_SET_SIZE, 10-16
EW_SNUGGLE_INSERT, 10-19
EW_TAB_KEY, 10-24
EWLS_EXTRACT, 21-2
EWLS_INIT, 21-2
FL_LIST_COMPLETE, 14-12
FL_LOCCHG, 14-12
FN_LIST, 13-2
FNS_EXTRACT_TAGS, 12-18
FNS_INSERT_TAGS, 12-18
FNS_SUBSET, 12-18
FNS_VALIDATE_FLIST, 12-18
FS_FSCAN, 13-4
GT_CHECK_ATS_ON, 23-5
GT_LAST_KEY, 23-5
h2LineConfirm, 24-10
HAI_KEY, 17-3, 17-4
hAppendEllipsis, 24-6
hAtob, 24-5
hAtos, 24-6
hBeep, 24-9
hBusyPrint, 24-9
hConfirm, 24-10
hDestroy, 24-2
hDlgItemDim, 24-21
hDlgItemLock, 24-21
hDlgSense, 24-17
hDlgSenseChlist, 24-18
hDlgSenseDtedit, 24-20
hDlgSenseEdwin, 24-17
hDlgSenseFledit, 24-19
hDlgSenseLledit, 24-19
hDlgSenseNcedit, 24-18
hDlgSensePtedit, 24-20
hDlgSenseRgedit, 24-19
hDlgSenseTwips, 24-22
hDlgSet, 24-11
hDlgSetChlist, 24-13
hDlgSetChlistOn, 24-13
hDlgSetDtedit, 24-15
hDlgSetEdwin, 24-12
hDlgSetFledit, 24-15
hDlgSetLledit, 24-14
hDlgSetNcedit, 24-14
hDlgSetPrompt, 24-12
hDlgSetPtedit, 24-15
hDlgSetRgedit, 24-16
hDlgSetText, 24-12
hDlgSetTitleByRid, 24-16
hDlgSetTwips, 24-22
hDlgTakeFocus, 24-21
hEnsurePath, 24-3
hErrorDialog, 24-11

hErrs, 24-4
hGetBBWid, 24-8
hGetBWid, 24-7
hGetSWid, 24-8
hInfoPrint, 24-8
hInfoPrintErr, 24-9
hInitVis, 24-2
hLaunchDial, 24-10
hLoadChlistResBuf, 24-4
hLoadResBuf, 24-4
hLoadResource, 24-3
hSetGFont, 24-7
hSetGStyle, 24-7
hSetGTmode, 24-6
hWservComSend, 24-2
IM_INIT, 20-3, 20-5, 20-8
IM_KEY, 20-2
IM_SENSE_BUF, 20-3, 20-5
IM_SENSE_VAL, 20-3, 20-5
IM_SET_BUF, 20-3, 20-5
IM_SET_RANGE, 20-6
IM_SET_VAL, 20-3, 20-5, 20-8
IM_TRANSITION, 20-3, 20-6, 20-9
IM_TRY_MATCH, 20-3, 20-6, 20-9
LB_DRAW_EMPHASIS, 6-10, 6-18, 6-19, 18-8
LB_DRAW_ITEM, 6-9, 6-17, 14-10, 18-8
LB_INQUIRE_FOCUS, 6-10
LB_INQUIRE_ITEM, 6-10, 14-11, 18-7
LB_INQUIRE_LAST, 6-11
LB_ITEM_WIDTH, 6-10, 6-18, 18-7
LB_SIZE_WINDOW, 6-9
LB_TAKE_FOCUS, 6-10
LG_DRAW, 5-12
LG_SELF_CHECK, 5-13, 9-6, 9-27, 10-29, 12-5, 12-10, 12-17
LG_SENSE_WIDTH, 5-13, 8-7, 8-12, 8-15, 8-21, 9-6, 10-16, 10-30, 11-3, 12-17, 17-8, 20-14
LG_SET_ID_POS, 5-12, 8-11, 10-16
LG_UPDATE, 5-13, 12-11, 12-17
LPR_INIT, 16-7
LPR_READ, 16-7
LPR_SENSE_BUF_WIDTH, 16-8
LPR_SENSE_TEXT, 16-8
LS_FILENAME, 16-44, 19-7
LS_SCAN, 19-7
MB_ADD_MENU, 6-16
MF_RANGE_BEEP, 9-7
p_false, 24-2
p_true, 24-2
PRINTING_DO_PRINT, 16-42
PRINTING_DONE, 16-42
PRINTING_SET_TITLE, 16-42
PS_NEW_LIST, 13-4
PT_COMPLETE, 19-4
PT_GETLINE, 19-3
PT_START, 19-3
SV_INIT, 19-3
SV_RUN, 19-3
TO_GET_SYSDAT, 22-3
TO_SET, 22-2
TO_SET_ABBREVIATIONS, 22-4
TO_SET_FORMAT, 22-2
top-level windows, 5-5
VA_COMPRESS, 8-25
VA_COPY, 8-24
VA_DELETEM, 8-24
VA_INIT, 8-24, 12-2
VA_INSERTM, 8-25
VA_PBUF, 8-25, 8-27, 12-2
VA_PREC, 8-25
VA_RECLEN, 8-24
VA_SEARCH, 12-2
VAN_SET_MAX, 8-27
WLD_RESTRICT, 20-14
WN_CALC_POSITION, 5-6
    and WN_POSITION, 5-7
WN_CONNECT, 5-5, 6-19
    and WN_POSITION, 5-7
WN_DODRAW, 5-5
WN_DRAW, 5-8, 5-9, 6-8, 6-15, 6-19, 6-20, 7-17, 8-5, 8-11, 8-15, 8-20, 9-6, 10-15, 10-31, 11-3, 17-7, 20-12
WN_EMPHASISE, 5-6, 5-10, 6-8, 7-20, 8-7, 8-21, 9-6, 9-27, 10-16, 10-31, 12-10, 18-10, 20-14
WN_INIT, 5-7, 5-12, 6-6, 6-14, 6-20, 8-5, 8-10, 8-15, 8-19, 8-22, 9-14, 9-17, 9-20, 9-25, 9-30, 9-33, 10-6, 10-27, 11-2, 12-4, 12-8, 12-14, 14-6, 17-7, 18-6, 18-10, 20-12
WN_KEY, 5-7, 6-7, 6-15, 6-17, 7-15, 8-6, 8-11, 8-19, 9-5, 9-27, 10-10, 10-25, 10-30, 12-5, 12-9, 12-16, 14-7, 17-8, 18-7, 18-10, 20-13
WN_POSITION, 5-7
WN_REDRAW, 5-5
WN_SENSE, 5-8, 7-14, 8-6, 8-11, 8-21, 8-23, 9-15, 9-18, 9-21, 9-26, 9-31, 9-33, 10-15, 10-28, 10-31, 12-9, 12-16, 14-10, 20-13
WN_SENSE_HELP, 5-6, 7-20, 10-15
WN_SET, 5-8, 7-14, 8-5, 8-21, 8-23, 9-6, 9-15, 9-18, 9-21, 9-26, 9-30, 9-33, 10-15, 10-28, 10-31, 11-3, 11-5, 12-5, 12-9, 12-14, 20-12
WN_VISIBLE, 5-6, 5-12, 6-14, 18-10
WS_ADD_DIAL, 3-15
WS_ADD_FILELIST, 3-22
WS_ALERT, 3-21
WS_ANIM_TICK, 3-23
WS_APPEND_COUNTRY, 3-21
WS_ATTACH_APP, 3-26
WS_BACKGROUND, 3-23
WS_CANCEL, 3-10
WS_CHANGE_CLIWIN, 3-14
WS_DATE_CHANGED, 3-24
WS_DEFINE_FNBAR, 3-24
WS_DIAL_ENV, 3-20
WS_DO_DIAL, 3-15
WS_DO_HELP, 1-2, 3-16
WS_DO_REMOTE_DIAL, 3-27
WS_DO_SUBMENU, 3-18
WS_DYN_INIT, 3-9
WS_EDIT_PDEV_SETUP, 3-22
WS_EDIT_PRINT_CONTEXT, 3-22
WS_ENS_PRINT_CONTEXT, 3-22
WS_ERROR_DIALOG, 3-19
WS_EVAL_ENV, 3-19
WS_EVALUATE, 3-19
WS_FILE_INFO_PRINT, 3-27
WS_FOREGROUND, 3-23
WS_FORMAT_DIALOG, 3-20
WS_FREE_DIAL, 3-17
WS_GET_ALLOC_INFO, 3-25
WS_GET_PRINT_CONTEXT, 3-25

WS_HIDE_APP, 3-26
WS_LAUNCH_DYL, 3-24
WS_LOAD_CHLIST_RES, 3-16
WS_LOCK, 3-17
WS_PROCESS_KEY, 3-12
WS_QUERY_DIALOG, 3-18
WS_REMOVE_DIAL, 3-16
WS_REMOVE_FILELIST, 3-23
WS_RESET_MENUBAR, 3-17
WS_RUN_MEMO, 3-27
WS_SELF_CHECK, 3-26
WS_SENSE_ACCEL, 3-15
WS_SENSE_PDEV_TEXT, 3-22
WS_SET_MENUBAR, 3-17
WS_SMART_DIAL, 3-21
WS_SWITCH_FILES, 3-17
WS_UNKNOWN_WM, 3-23
WS_USER_ABANDONED, 3-28
WS_WRAP_PARA, 3-16