

SIBO 'C' Software Development Kit

XADD 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, Psion MC, Psion HC, Psion Series 3, Psion Series 3a and Psion *Workabout* 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 XADD 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 Print Preview Classes	2-1
Precursors.....	2-1
Class diagram	2-2
LPRINTER.....	2-2
Class definition.....	2-3
Property.....	2-3
PDEVDLG methods.....	2-3
Destroy	2-3
Initialise.....	2-3
PRVVIEW.....	2-4
Class definition.....	2-5
Property.....	2-6
Resources	2-10
PRVVIEW methods	2-10
Destroy	2-10
Handle keypress	2-11
Draw.....	2-12
Set display mode	2-12
Initialise.....	2-12
Completion call-back	2-14
Completion call-back	2-15
Toggle margins	2-15
Move to page.....	2-15
Initialise.....	2-15
PRVINFO	2-16
Class definition.....	2-16
Property.....	2-17
Resources	2-17
PRVINFO methods	2-17
Initialise.....	2-17
Set number of pages.....	2-17
Draw.....	2-17

PRVPAGE.....	2-18
Class definition.....	2-19
Property	2-19
Resources	2-19
PRVPAGE methods	2-20
Dodraw.....	2-20
Draw.....	2-20
Initialise.....	2-21
Margins	2-21
PRVCOMM	2-21
Class definition.....	2-22
Property	2-22
Resources	2-22
PRVCOMM methods	2-22
Initialise.....	2-22
Set menu item text.....	2-22
Exit application	2-23
Print.....	2-23
Toggle margins	2-23
Launch Preview options dialog.....	2-23
Launch Jump to page dialog	2-23
Exit print preview.....	2-24
PRVOPTIONS_DLG	2-24
Class definition.....	2-24
Property.....	2-25
Resources	2-25
PRVOPTIONS_DLG methods.....	2-25
Dynamically initialise	2-25
Handle key input.....	2-25
PRVJUMP_DLG.....	2-26
Class definition.....	2-26
Property.....	2-26
Resources	2-27
PRVJUMP_DLG methods	2-27
Dynamically initialise	2-27
Handle key input.....	2-27

3 Calendar Classes.....	3-1
Precursors.....	3-1
Class diagram	3-1
CALIMGWN	3-2
Class definition.....	3-2
Property	3-2
CALIMGWN methods	3-2
Redraw the calendar	3-2
CALWIN.....	3-3
Class definition.....	3-4
Property	3-4
CALWIN methods	3-4
Destroy the calendar.....	3-4
Initialise the calendar	3-5
Handle a keypress	3-7
Sense current date	3-9
Emphasise	3-9
Examples	3-10
Creating a CALWIN component.....	3-10
Handling keypresses.....	3-10

4 Automatic Test System Classes	4-1
Precursors	4-1
Class diagram	4-1
ATSSV	4-2
Class definition	4-2
Property	4-3
Auxiliary structures	4-3
ATSSV methods	4-4
Initialise	4-4
Process an ATS message	4-4
Process an error	4-6
Free inter-process message	4-6
Report a keypress	4-6
ATSTIM	4-7
Class definition	4-7
Property	4-7
ATSTIM methods	4-7
Initialise	4-7
Process completion	4-7

5 Additional Active Object Classes	5-1
Precursors	5-1
Class diagram	5-1
LOGONA	5-2
Class definition	5-2
Property	5-2
LOGONA methods	5-2
Initialise	5-2
Queue request	5-3
Cancel request	5-3
Report process termination	5-3
UNLOAD	5-3
Class definition	5-4
Property	5-4
UNLOAD methods	5-4
Destroy	5-4
Initialise	5-4
Process completion	5-4

CHAPTER 1

INTRODUCTION

This manual is a reference document for Psion's XADD library. It provides a 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 *XADD Reference* manual.

The XADD library is supplied as the *xadd.dyl* dynamic link library in the ROM of the Series 3a and *Workabout* machines. *It is not present on Series 3 machines.*

Because of their different screen sizes, the Series 3a and *Workabout* machines contain slightly differing versions of the XADD library. The only difference, however, is in the implementation of the *CALWIN* calendar window class. On the Series 3a, this can be set to display one, three or twelve months, but on the *Workabout* the display is restricted to a single month.

XADD contains classes that extend the functionality of the HWIM and FORM user interface, notably to provide print preview facilities.

Some classes in the XADD library are designed solely for internal use by system code and are not documented in this manual. The undocumented classes are:

OPRINTER	Implements WDR printing for OPL.
OHELPDLG	Implements Help dialogs for OPL.
RUNMEMO	Implements Memo editing from the Agenda application.
IPCSTAT	Used for inter-process communication of status data during Memo editing.

Some of the documented XADD 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 XADD classes

An application (or DYL) that either subclasses or creates an instance of an XADD class must declare an external reference to the XADD 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 xadd
```

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

Since the XADD library contains extensions to FORM and HWIM, it is likely that access will also be needed to these libraries, so the category file will also need to contain `EXTERNAL` references to one or both of the FORM and HWIM libraries.

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

```
p_new(CAT_MYPROG_HWIM, C_LOGONA);
```

If *myprog.cat* defines a subclass of an XADD class (say, the class SUBLOGONA) 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_SUBLOGONA);
```

Similar considerations 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 `PRVVIEW`.

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 `PRVVIEW` with the title line:

PVV_DONE Completion call-back

and prototyped as:

```
INT pvv_done(INT event);
```

would be invoked by, for example:

```
p_send3(hand, O_PVV_DONE, PAGES_DONE_PAGE);
```

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 INT prview_pvv_done(PR_WSERV *self, INT event)
{
    ...
}
```

The $\star$ 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 $\star$ 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 is 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.

XADD 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

PRINT PREVIEW CLASSES

The classes described in this chapter are as follows:

- the `XPRINTER` class which provides the basic framework for both printing and print preview operations. The `XPRINTER` class must be subclassed to be useful.
- the `PRVVIEW` class which implements print preview operations. A `PRVVIEW` object is used as a component by the `XPRINTER` class.
- the `PRVINFO` class which implements an information window indicating the number of pages being print previewed. A `PRVINFO` object is used as a component by the `PRVVIEW` class.
- the `PRVPAGE` class which decompresses a page image stored in an external memory segment and then draws the page image in a print preview window. Multiple `PRVPAGE` objects are used as components by the `PRVVIEW` class.
- the `PRVCOMM` class which provides the print preview command manager. The print preview command manager is installed by the `PRVVIEW` class.
- the `PRVOPTIONS_DLG` class which implements the *Preview options* dialog: this allows the user to select the number of pages displayed during the print preview operation. A *Preview options* dialog is used by the `PRVCOMM` class.
- the `PRVJUMP_DLG` class which implements the *Jump to page* dialog: this allows the user to select the first page displayed on screen during the print preview operation. A *Jump to page* dialog is used by the `PRVCOMM` class.

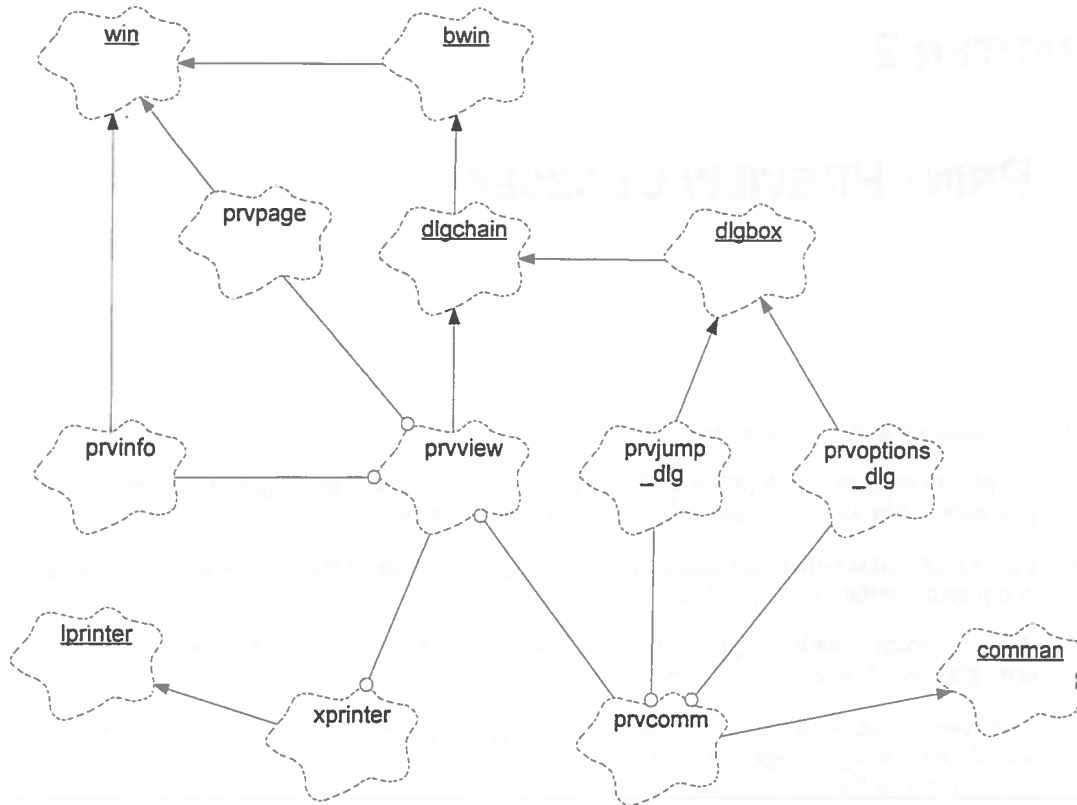
Note that an application would normally only create an instance of the `XPRINTER` class (and perhaps exceptionally, of the `PRVVIEW` class): the other component classes are intimately associated with the `PRVVIEW` class and are thus not expected to be either directly instanced or subclassed.

Precursors

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

- the `LPRINTER` class described in the *Print Classes* chapter of the *HWIM Reference* manual.
- the `PAGES` class described in the *Document Printing Classes* chapter of the *FORM Reference* manual.
- the `PRVPDR` class described in the *Print Preview Class* chapter of the *FORM Reference* manual.
- the `DLGBOX` class described in the *Dialog Boxes* chapter of the *HWIM Reference* manual.

Class diagram



XPRINTER

LPRINTER	XPRINTER
pages wdr f defer wtab lheight width subsqind	
destroy lpr_init lpr_read lpr_sense_buf_width lpr_sense_text	destroy lpr_init

The XPRINTER class provides the basic framework that allows an application to perform both printing and print preview operations using the services provided by the PRINTER and PAGES classes.

The XPRINTER class provides a deferred method - the `lpr_sense_text` method - which is supplied as a callback to the PAGES active object: this method must be replaced so that it obtains the next fragment of text to print.

For further details of printing see the LPRINTER class in the *Print Classes* chapter of the *HWIM Reference* manual and the *Printing* chapter in the *Object Oriented Programming* manual.

Class definition

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

```

CLASS xprinter lprinter
{
  REPLACE destroy
  REPLACE lpr_init
  PROPERTY
  {
    WORD locked;
  }
}

```

Property

`xprinter.locked` TRUE if an extra level of locking has been added to the application by the XPRINTER object, otherwise FALSE.

XPRINTER methods**★ DESTROY****Destroy**

VOID destroy(VOID);

Destroy the XPRINTER object.

If `xprinter.locked` is non-zero, removes one level of locking by sending a `WS_LOCK` message to `w_ws` with an argument of FALSE.

Supersends a DESTROY message.

LPR_INIT**Initialise**

VOID lpr_init(INT commid);

Initialise the XPRINTER component and, if `commid` is zero, start a printing operation, otherwise start a print preview operation.

If `commid` is zero, adds a level of locking by sending a `WS_LOCK` message to `w_ws` with an argument of TRUE and then writing TRUE to `xprinter.locked`.

Either starts a printing operation, or initialises a print preview operation, by writing `commid` to `lprinter.subsqind` and then supersending an LPR_INIT message.

If `commid` is zero, returns.

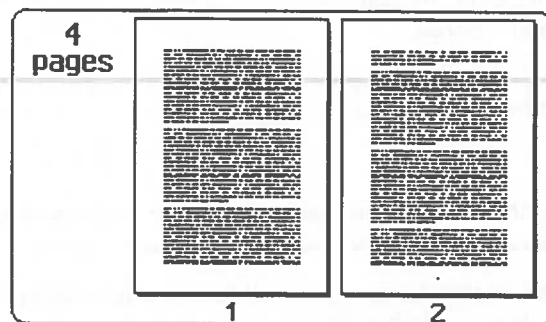
Otherwise creates an instance of the PRVVIEW class and initialises the PRVVIEW object by sending a `PVV_INIT` message with an argument of `self`, `commid` and the address of `lprinter.pages`:

- the `self` argument specifies that the `lpr_read` call-back method required by the `PAGES` active object is to be supplied by the XPRINTER object.
- the `commid` argument should specify the method number of a print method in the current application manager.
- the address of `lprinter.pages` is supplied so that it may receive the handle of the `PAGES` active object which is handling the print preview.

PRVVIEW

WIN	BWIN	DLGCHAIN	PRVVIEW
flags id		next	pPages pPrvInfo pPageView pOldMenu pOldComman Started PrintMethod hdone mdone NoPageViews MaxNoPageViews ScrollArea Pv
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_draw wn_set wn_init wn_sense_help pvv_done pvv_pages_done pvv_new_page pvv_margins pvv_init pvv_false

The PRVVIEW class supports the print preview of one or more pages in a document. An example print preview display is shown in the following picture:



The number of pages that may be displayed is determined by the dimensions of the screen up to a limit of PVV_MAX_PAGEVIEWS. The page images are drawn to the screen by an array of PRVPAGE components.

The text in the top left of the screen is an information window that indicates the number of pages in the preview range. This information window is provided by a PRVINFO component.

The print preview is carried out by means of PAGES and PRVPDR objects. These treat the pages as a sequence of fragments or print elements each of which contains no more than one line of text.

The print elements are passed to the PAGES object by means of a so-called read-call-back: when the PAGES object requires the next print element it sends an LPR_READ message to the supplier object. The handle of the supplier object is passed to the PAGES object on initialisation and in turn must be passed to the PRVVIEW object on its initialisation. This object is usually an instance of an XPRINTER subclass.

The PRVVIEW object is notified by the PAGES object of the completion of each stage of the print preview by means of a so-called completion call-back: when the next stage is complete the PAGES object simply sends a

pvv_pages_done message to the prview object. This message is only significant when the preview is about to terminate.

The prview object is also notified by the prvpdr object of the completion of each stage of the print preview by means of a completion call-back: when the next stage is complete the prvpdr object simply sends pvv_done messages to the prview object. In this case the message updates the page count and then updates the display to include any newly created page image.

The PAGES class is described in the *Document Printing Classes* chapter of the *FORM Reference* manual.

The PRVPDR class is described in the *Print Preview Class* chapter of the *FORM Reference* manual.

It is assumed that the global variable `datGate` specifies the handle of an instance of the GATE class. In the description of the methods this GATE instance is simply referred to as the GATE object.

It is also assumed that `w->am.appman.spare1` contains the handle of an instance of the PRINTER class: the PRINTER class automatically writes its handle to `w->am.appman.spare1` on initialisation.

Class definition

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

```

CLASS prview dlchain
{
  REPLACE destroy
  REPLACE wn_key
  REPLACE wn_draw
  REPLACE wn_set
  REPLACE wn_init
  REPLACE wn_sense_help
  ADD pvv_done
  ADD pvv_pages_done
  ADD pvv_new_page
  ADD pvv_margins
  ADD pvv_init
  ADD pvv_false=p_false

  CONSTANTS
  {
    PVV_INTERPAGE_GAP 6
    PVV_TOP_GAP 4
    PVV_FOOTER_GAP 12
    PVV_PAGE_SHADOW 1
    PVV_SIDE_GAP 12
    PVV_PAGENO_DISP_GAP 60
    PVV_DISP_FACING 0
    PVV_DISP1 1
    PVV_DISP2 2
    PVV_DISP3 3
    PVV_DISP4 4
    PVV_MARGINS_ON 0x01
    PVV_PREVIEW_SET 0x80
    PVV_VIEW_MAX_PAGEVIEWS 4
  }
}

```

```

TYPES
{
    typedef struct
    {
        PAGES_CALLS Calls;
        WORD PrintMethod;
        WORD Spare1;
        WORD Spare2;
    } IN_PRVVIEW;

    typedef struct
    {
        P_RECT Margins;      margin area
        INT HeadTop;
        INT HeadBot;
        P_RECT Border;      area in which to draw greeked page
        P_RECT Footer;      area in which to print page number
    } PVV_PAGE_DATA;

    typedef struct
    {
        PR_ROOT *pArray;    varray containing page index into preview segment
        INT NoPages;        number of pages (so far)
        INT FirstPage;      zero for FACING_PAGES, one otherwise
        INT LastPage;       =NoPages if not FACING_PAGES, or rounded up to odd number
        INT PageOffset;     page number of first page displayed
        INT PageNo;         current page no
        PVV_DISPLAY Disp;
        INT UseGrey;
        INT LeftOffset;
        INT PageViewWidth;  width of PageView (pixels)
        PVV_PAGE_DATA Page;

! Following are scratch areas used by all pageviews
        UBYTE *pBitRow;     current row from bitmap
        UBYTE *pLastRow;    previous row from bitmap
        BMP_RASTER_ROW_REC *pRowRec;    compressed data from preview segment
        HANDLE PrvSegHandle;    handle of preveiw segment
        PRV_BITMAP Bmp;        bitmap, loaded from preview segment
    } PVV_PAGEVIEW_DATA;
}

PROPERTY 6
{
    PR_PAGES *pPages;      Pages active object, destroy on error
    PR_PRVINFO *pPrvInfo;  Info window
    PR_ROOT *pPageView[PVV_VIEW_MAX_PAGEVIEWS]; PageView objects
    WSERV_INFO *pOldMenu;  holds ptr of old Menu when in preview mode
    PR_COMMAND *pOldComman; holds ptr of old Comman when in preview mode
    WORD Started;          set if am_start called
    WORD PrintMethod;      command manager print method
    VOID *hdone;           Callback handle for %done & completion
    WORD mdone;            Callback method for %done & completion
    INT NoPageViews;       number of PageViews on display
    INT MaxNoPageViews;    max no of PageViews that will fit in view
    P_RECT ScrollArea;     area in which pages are displayed
    PVV_PAGEVIEW_DATA Pv;  property that can be peeked by constituent pageviews
}
}

```

Property

prvview.pPages	The handle of a PAGES active object, used to create compressed page images for all pages in the print preview range. For details see the <i>Document Printing Classes</i> chapter of the <i>FORM Reference</i> manual.
prvview.pPrvInfo	The handle of a PRVINFO object that provides an information window indicating the total number of pages in the print preview range.

<code>prvview.pPageView</code>	An array of up to <code>PVV_MAX_PAGEVIEWS</code> elements, each of which may be a <code>PRVPAGE</code> object handle. The <code>PRVPAGE</code> objects are used to draw the page images in the print preview window - the page images are read from an external memory segment whose handle is stored in <code>prvview.Pv.PrvSegHandle</code> .
On initialisation of each <code>PRVPAGE</code> object:	
	the <code>win.id</code> property is set to that of the parent <code>PRVVIEW</code> object,
	the <code>prvpage.init.Pos</code> property is set to the index of the object in the <code>prvview.pPageView</code> array,
	the <code>prvpage.init.pPrvView</code> property is set to the handle of the parent <code>PRVVIEW</code> object.
<code>prvview.pOldMenu</code>	The address of a memory cell containing the original command manager menu data: this is organised as a <code>WSERV_INFO</code> resource struct.
<code>prvview.pOldComman</code>	The handle of the original command manager of the application: the <code>PRVVIEW</code> class installs a print preview command manager on initialisation.
<code>prvview.Started</code>	Set to <code>TRUE</code> if an <code>AM_START</code> message was sent to <code>w_am</code> on initialisation.
<code>prvview.PrintMethod</code>	The method number of a print method in the original command manager: it is used to allow printing during print preview operations.
<code>prvview.hdone</code>	The handle of the object to which the <code>PVV_PAGES_DONE</code> method sends a <code>prvview.hdone</code> message on completion of either a page or the document.
<code>prvview.mdone</code>	The method number of the message sent by the <code>PVV_PAGES_DONE</code> method on completion of either a page or the document.
<code>prvview.NoPageViews</code>	The number of page images displayed in the print preview window.
<code>prvview.MaxNoPageViews</code>	The maximum number of page images that may be displayed in the print preview window. It is set to four on initialisation.
<code>prvview.ScrollArea</code>	Specifies a rectangle enclosing the page views and the page numbers.
<code>prvview.Pv</code>	Contains information about the page views, as described below.

The `PVV_PAGEVIEW_DATA` struct is defined as follows:

```
typedef struct
{
    PR_ROOT *pArray;
    INT NoPages;
    INT FirstPage;
    INT LastPage;
    INT PageOffset;
    INT PageNo;
    PVV_DISPLAY Disp;
    INT UseGrey;
    INT LeftOffset;
    INT PageViewWidth;
    PVV_PAGE_DATA Page;
    UBYTE *pBitRow;
    UBYTE *pLastRow;
    BMP_RASTER_ROW_REC *pRowRec;
    HANDLE PrvSegHandle;
    PRV_BITMAP Bmp;
} PVV_PAGEVIEW_DATA;
```

The significance of the members of the `PVV_PAGEVIEW_DATA` struct is as follows:

<code>pArray</code>	The handle of a <code>VAFLAT</code> object, used to store the offsets in an external memory segment of the compressed page images: the memory segment handle is stored in <code>prvview.Pv.PrvSegHandle</code> .
---------------------	--

Each offset is stored as `LONG` data.

NoPages	The number of pages in the print preview range: may be less than the number of pages in the document.												
FirstPage	For internal use only.												
LastPage	For internal use only.												
PageOffset	The page offset of the first page in the print preview range. It is zero when the preview range starts from the first page in the document.												
PageNo	The page offset of the first page <i>displayed</i> in the print preview window. The offset is with respect to the first page in the preview range and is thus zero when the first page is visible.												
Disp	The current print preview settings. The PVV_DISPLAY struct is defined as follows: <pre>typedef struct { UBYTE Mode; UBYTE Flags; } PVV_DISPLAY;</pre> the Mode member may contain <i>one</i> of the following: <table><tr><td>PVV_DISP_FACING</td><td>specifies that an odd page is to be displayed first followed by an even page.</td></tr><tr><td>PVV_DISP1</td><td>specifies that one page is to be displayed.</td></tr><tr><td>PVV_DISP2</td><td>specifies that two pages are to be displayed.</td></tr><tr><td>PVV_DISP3</td><td>specifies that three pages are to be displayed.</td></tr><tr><td>PVV_DISP4</td><td>specifies that four pages are to be displayed.</td></tr></table> the Flags member may contain: <table><tr><td>PVV_MARGINS_ON</td><td>specifies that margins are to be visible during print preview operations.</td></tr></table>	PVV_DISP_FACING	specifies that an odd page is to be displayed first followed by an even page.	PVV_DISP1	specifies that one page is to be displayed.	PVV_DISP2	specifies that two pages are to be displayed.	PVV_DISP3	specifies that three pages are to be displayed.	PVV_DISP4	specifies that four pages are to be displayed.	PVV_MARGINS_ON	specifies that margins are to be visible during print preview operations.
PVV_DISP_FACING	specifies that an odd page is to be displayed first followed by an even page.												
PVV_DISP1	specifies that one page is to be displayed.												
PVV_DISP2	specifies that two pages are to be displayed.												
PVV_DISP3	specifies that three pages are to be displayed.												
PVV_DISP4	specifies that four pages are to be displayed.												
PVV_MARGINS_ON	specifies that margins are to be visible during print preview operations.												
UseGrey	set to TRUE to specify that the grey plane may be used, set to FALSE otherwise.												
LeftOffset	specifies the horizontal separation of the left edge of the first page view from the left edge of the window.												
PageViewWidth	specifies the width of a page view in units of pixels.												

Page specifies the layout of a generic page view on the screen: this generic page view has zero horizontal offset from the left edge of the main window.

The `PVV_PAGE_DATA` struct is defined as follows:

```
typedef struct
{
    P_RECT Margins;
    INT HeadTop;
    INT HeadBot;
    P_RECT Border;
    P_RECT Footer;
} PVV_PAGE_DATA;
```

Margins - specifies the rectangle that is enclosed by the page margins.

HeadTop - specifies the distance of the header text from the top margin.

HeadBot - specifies the distance of the footer text from the bottom margin.

Border - specifies a rectangle enclosing the page view i.e. the physical page border.

Footer - specifies a rectangle enclosing the page number.

Note: in all cases the units are pixels.

pBitRow for internal use.

pLastRow for internal use.

pRowRec for internal use.

PrvSegHandle specifies the handle of an external memory segment that is used to store compressed page images.

The memory segment is created and written to by `PR_PREVIEW_START` and `PR_PREVIEW` messages that are sent on initialisation.

Bmp contains information about an external bitmap that is used internally when transferring compressed page images to the print preview window.

The `PRV_BITMAP` struct is defined as follows:

```
typedef struct
{
    INT Id;
    HANDLE SegHandle;
    UPOINT Size;
    UWORD ByteWidth;
    UWORD BitWidth;
} PRV_BITMAP;
```

The significance of the members of the above struct is as follows:

Id specifies the ID of the bitmap.

SegHandle specifies the handle of a memory segment that contains the bitmap.

Size the *x* member specifies the width of the bitmap in units of pixels.

the *y* member specifies the height of the bitmap in units of pixels.

ByteWidth specifies the number of bytes required to store one horizontal line from the bitmap i.e. $\text{size.x} \div 8 + 1$.

BitWidth specifies the number of horizontal bits needed to draw a page in the correct proportion.

Resources

Defined in the system resource file *sx_ra*.

```
RESOURCE WSERV_INFO sys_preview_acc
{
    menubar_id=sys_preview_menubar;
    first_com=O_PVC_PREVIEW_PRINT;
    accel=
    {
        'p', /* Print */
        'm', /* Margins */
        'd', /* Pages to display */
        'j', /* Jump to page */
        '*' /* Exit preview */
    };
}
```

PRVVIEW methods

★ WN_SENSE_HELP

Sense print help resource

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

Sense the ID of the print help resource.

The method simply returns `-SYS_HELP_PRINT`.

★ DESTROY

Destroy

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

Free any resources and destroy the PRVVIEW object.

Cancels any busy messages.

If `prvview.pOldMenu` is non-zero, resets the application's menu bar by sending a `WS_RESET_MENUBAR` message to `w_ws` specifying an argument of `prvview.pOldMenu`.

If `prvview.pOldComman`, destroys the print preview command manager by sending a `DESTROY` message to `w_ws->wserv.com` and then writes `prvview.pOldComman` to `w_ws->wserv.com`.

Terminates the print preview by sending a `PR_PREVIEW_END` message to `w_am->appman.spare1`.

Frees the bitmap specified by `prvview.Pv.Bmp.Id`.

If `prvview.Pv.Bmp.SegHandle` is non-zero, closes the memory segment with handle `prvview.Pv.Bmp.SegHandle`.

Frees the memory cell at address `prvview.Pv.pBitRow`.

If `prvview.Pv.pArray` is non-zero, sends a `DESTROY` message to `prvview.Pv.pArray`.

If `win.flags` contains `PR_WIN_INITIALISED`, sends a `WS_REMOVE_DIAL` message to `w_ws`.

If `prvview.Started` is non-zero:

- sends self a `DESTROY` message.
- sends an `AM_STOP` message to `w_am`.

Otherwise:

- sends self a `DESTROY` message.

WN_KEY**Handle a keypress**

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

Handle a keypress that might terminate the print preview, or scroll the page view.

If keycode is `W_KEY_ESCAPE`:

- if `prvview.pPages` is non-zero, sends a `DESTROY` message to `prvview.pPages`.
- writes `NULL` to `prvview.pPages`.
- returns `TRUE`.

If keycode is `W_KEY_UP`:

- attempts to scroll backwards `prvview.NoPageViews` pages by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` minus `prvview.NoPageViews`.

If keycode is `W_KEY_DOWN`:

- attempts to scroll forwards `prvview.NoPageViews` pages by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` plus `prvview.NoPageViews`.

If keycode is `W_KEY_PAGE_DOWN`:

- if `modifiers` contains `W_CTRL_MODIFIER`, attempts to display the last pages in the allowed preview range by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.LastPage` minus `prvview.NoPageViews` plus one.
- otherwise, if `prvview.Pv.Disp.Mode` contains `PVV_DISP_FACING`, attempts to move forwards by `prvview.Pv.PageNo` pages by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` plus `prvview.NoPageViews`.
- otherwise, attempts to moves forward one page by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` plus one.

If keycode is `W_KEY_RIGHT`:

- if `prvview.Pv.Disp.Mode` contains `PVV_DISP_FACING`, attempts to move forwards by `prvview.Pv.PageNo` pages by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` plus `prvview.NoPageViews`.
- otherwise, attempts to moves forward one page by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` plus one.

If keycode is `W_KEY_PAGE_UP`:

- if `modifiers` contains `W_CTRL_MODIFIER`, attempts to display the first pages in the allowed preview range by sending self a `PVV_NEW_PAGE` message with as argument zero.
- otherwise, if `prvview.Pv.Disp.Mode` contains `PVV_DISP_FACING`, attempts to move forwards by `prvview.Pv.PageNo` pages by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` minus `prvview.NoPageViews`.
- otherwise, attempts to moves backwards one page by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` minus one.

If keycode is `W_KEY_LEFT`:

- if `prvview.Pv.Disp.Mode` contains `PVV_DISP_FACING`, attempts to move forwards by `prvview.Pv.PageNo` pages by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` minus `prvview.NoPageViews`.
- otherwise, attempts to moves backwards one page by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.PageNo` minus one.

If keycode is `W_KEY_HOME`:

- moves to the first page(s) in the preview range by sending self a `PVV_NEW_PAGE` message with as argument zero.

If keycode is `W_KEY_END`:

- moves to the last pages in the preview range by sending self a `PVV_NEW_PAGE` message with as argument `prvview.Pv.LastPage` minus `prvview.NoPageViews` plus one.

Returns `FALSE`.

WN_DRAW

Draw

`VOID wn_draw(VOID);`

Draw the page views on screen.

Draws a border by calling `gBorder` with as argument `win.flags`.

Sends a `WN_DRAW` message to each `PRVPAGE` object - the handles of the `PRVPAGE` objects are stored in the first `prvview.NoPageViews` elements of `prvview.pPageView`.

WN_SET

Set display mode

`VOID wn_set(INT mode);`

Set the number of pages to display during the print preview as specified by `mode`.

The allowed values of `mode` are as follows:

<code>PVV_DISP_FACING</code>	specifies that two facing pages are to be displayed i.e. the first page must always have an odd page number.
<code>PVV_DISP1</code>	specifies that one page is to be displayed.
<code>PVV_DISP2</code>	specifies that two pages are to be displayed.
<code>PVV_DISP3</code>	specifies that three pages are to be displayed.
<code>PVV_DISP4</code>	specifies that four pages are to be displayed.

If `mode` is equal to `prvview.Pv.Disp.Mode`, and thus the display mode is already set, returns.

Writes appropriate values into property including `prvview.NoPageViews`: note that if `mode` specifies more pages than can be displayed, `mode` is reset to the limit.

Writes `mode` to `prvview.Pv.Disp.Mode`.

Sets `PVV_PREVIEW_SET` in `prvview.Pv.Disp.Flags`.

Writes `prvview.Pv.Disp` to `DatGate->gate.prevdsp`.

If necessary the print preview window is resized whilst ensuring that it remains centred on screen.

WN_INIT

Initialise

`VOID wn_init(IN_PRVVIEW *pIn, INT DoAmStart);`

Initialise the `PRVVIEW` object and then start the print preview operation.

The `IN_PRVVIEW` struct is defined as follows:

```
typedef struct
{
    PAGES_CALLS Calls;
    WORD PrintMethod;
    WORD Spare1;
    WORD Spare2;
} IN_PRVVIEW;
```

The members of the `IN_PRVVIEW` struct have the following significance:

<code>Calls.hread</code>	specifies the handle of an object that is to be sent read call-back messages by the <code>PAGES</code> active object when it requires the next print element.
--------------------------	---

<code>Calls.mread</code>	specifies the read call-back message that is sent by the <code>PAGES</code> active object when it requires the next print element.
<code>Calls.hdone</code>	specifies the value that is written to <code>prvview.hdone</code> .
<code>Calls.mdone</code>	specifies the value that is written to <code>prvview.mdone</code> .
<code>PrintMethod</code>	specifies the value that is written to <code>prvview.PrintMethod</code> .
<code>Spare1</code>	this member is not used.
<code>Spare2</code>	this member is not used.

If the `gate.prevdisp.Flags` property of the `GATE` object is non-zero, writes the `gate.prevdisp` property to `prvview.Pv.Disp`.

Writes a default display mode to `prvview.Pv.Disp.Mode`: the default is `PVV_DISP2`,

If the `PSP` environment variable exists - this environment variable is four bytes long:

- subtracts '0' from the value in the first byte and writes the result to `prvview.Pv.Disp.Mode`.
- if `prvview.Pv.Disp.Mode` is greater than `PVV_DISP4`, resets `prvview.Pv.Disp.Mode` to `PVV_DISP_FACING`.
- if the second byte contains '1', writes `PVV_MARGINS_ON` to `prvview.Pv.Disp.Flags`.
- otherwise writes zero to `prvview.Pv.Disp.Flags`.

Writes `TRUE` to `prvview.Pv.UseGrey` and then connects to the window server by sending `self` a `WN_CONNECT` message. Sets the dimensions of the print preview window ensuring that it is centred on screen.

Creates an instance of the `PRVINFO` class and writes its handle to `prvview.pPrvInfo`. Connects the `PRVINFO` object to the window server and then initialises it.

Sets various items of property as follows:

- sets property associated with the page range: this property is `prvview.Pv.PageOffset`, `prvview.MaxNoPageViews`, `prvview.NoPageViews` and `prvview.Pv.FirstPage`.
- sets property associated with the layout of the page views: this property is `prvview.Pv.Page.Border`, `prvview.Pv.Page.Header`, `prvview.Pv.Page.Footer`, `prvview.Pv.Page.Margins`, `prvview.Pv.Page.HeadTop`, `prvview.Pv.Page.HeadBot`, `prvview.Pv.ScrollArea` and `prvview.ScrollArea`.
- sets the content of the `prvview.Pv.Bmp` property.

Creates an instance of the `VAFLAT` class and writes its handle to `prvview.Pv.pArray`. Initialises and sets the capacity of the `VAFLAT` component.

Initialises the current printer object for a print preview operation by sending a `PR_PREVIEW_START` message to `w_am->appman.spare1` with as argument the address of a `PREVIEW_INIT` struct initialised as follows:

<code>pSegName</code>	specifies a pointer to a buffer containing the name to be given to an external memory segment that will contain the page images. The name supplied is "PRV.ext" where the <i>ext</i> component is the extension in the name of the current process.
<code>pArray</code>	this is set to <code>prvview.Pv.pArray</code> .
<code>PrvSize</code>	set to <code>prvview.Pv.Bmp.Size</code> .
<code>BitWidth</code>	set to <code>prvview.Pv.Bmp.BitWidth</code> .
<code>hPrvDone</code>	set to <code>self</code> : specifies the handle of the object to which the <code>PRVPDR</code> object sends <code>mPrvDone</code> messages after the completion of each stage of the print preview operation.
<code>mPrvDone</code>	set to <code>O_PVV_DONE</code> : specifies the message to be sent by the <code>PRVPDR</code> object after completion of each stage of the print preview.

Writes the return value from the `PR_PREVIEW_START` message to `prvview.Pv.PrvSegHandle`.

Installs the print preview command manager as follows:

- sends a `WS_SET_MENUBAR` message to `w_ws` with an argument of `-SYS_PREVIEW_ACC` system resource and writes the address of the original menu to `prvview.pOldMenu`.
- creates an instance of the `PRVCOMM` class and writes its handle to `w_ws->wserv.com` and writes the handle of the original command manager to `prvview.pOldComman`.
- initialises the print preview command manager by sending a `COM_INIT` message to `w_ws->wserv.com` with an argument of `self`.

Initialises the page images as follows:

- creates and initialises `PVV_VIEW_MAX_PAGEVIEWS` instances of the `PRVPAGE` class and writes the handles to the `prvview.pPageView` array.
- draws the page images in the print preview window by sending a `WN_DODRAW` message to each element of `prvview.pPageView`.

Starts the print preview operation by sending a `PR_PREVIEW` message to `w_am->appman.spare1` with as argument the address of a `PAGES_CALLS` struct initialised as follows:

<code>hread</code>	set to <code>pIn->Calls.hread</code> : specifies the handle of an object that is to be sent read call-back messages by the <code>PAGES</code> active object when it requires the next print element.
<code>mread</code>	set to <code>pIn->Calls.mread</code> : specifies the read call-back message that is sent by the <code>PAGES</code> active object when it requires the next print element.
<code>hdone</code>	this is set to <code>self</code> : specifies the handle of an object that is to be sent an <code>mdone</code> message by the <code>PAGES</code> active object after the completion of each stage of the print preview operation.
<code>mdone</code>	this is set to <code>PVV_PAGES_DONE</code> . specifies the message that is to be sent the <code>PAGES</code> active object after the completion of each stage of the print preview operation.

Writes the handle of the current `PAGES` active object - i.e. the return value from the `PR_PREVIEW` message - to `prvview.pPages`.

Makes the print preview window visible by sending `self` a `WN_INITVIS` message.

Displays an information message containing the text in the `SYS_BUSY` system resource: on English language machines this is "Busy".

Adds the preview object as a dialog with an associated menu by sending a `WS_ADD_DIAL` message to `w_ws` with an argument of `self`.

Sets `PR_WIN_INITIALISED`, `DLGCHAIN_WIH_MENU` and `PR_WIN_NO_DDP` in `win.flags`.

If `DoAmStart` is non-zero:

- writes `TRUE` to `prvview.Started`.
- allows queued active objects to run by sending an `AM_START` message to `w_am`.

PVV_DONE

Completion call-back

```
INT pvv_done (INT event);
```

Complete processing of the current stage in the print preview operation: this call-back method is called by the `PRVPDR` object.

If `event` is `PAGES_DONE_PAGE` indicating that the current page image has been completed:

- increments the page count i.e. `prvview.Pv.NoPages` and updates `prvview.Pv.LastPage`.
- updates the number of pages in the preview display by sending a `WN_SET` message to `prvview.pPrvInfo` with an argument of `prvview.Pv.NoPages`.
- if the page is visible, then draws the page by sending a `WN_DODRAW` message to the appropriate element of the `prvview.pPageView` array.

Otherwise if `event` is either `PAGES_DONE_ERROR` or `PAGES_DONE_END`, cancels any busy message and writes `NULL` to `prvview.pPages`.

Returns TRUE.

PVV_PAGES_DONE

Completion call-back

```
INT pvv_pages_done(PAGES_DONE *d, PAGES_INIT *par);
```

Complete processing of the current stage in the print preview operation: this call-back method is called by the PAGES active object.

If d->event is either PAGES_DONE_ERROR or PAGES_DONE_END, the method simply cancels any busy message and then writes NULL to prview.pPages since the PAGES active object is about to destroy itself.

Otherwise if prview.hdone is non-zero, the method sends a prview.mdone message to prview.hdone with as arguments d and par and then returns the return value.

Otherwise the method returns FALSE.

Note: when d->event is equal to PAGES_DONE_PAGE indicating completion of the current page, it is not guaranteed that the page image is complete.

PVV_MARGINS

Toggle margins

```
VOID pvv_margins(VOID);
```

Toggle the visibility of the margins in the print preview display.

If prview.Pv.Disp.Flags contains PVV_MARGINS_ON, clears PVV_MARGINS_ON in prview.Pv.Disp.Flags.

Otherwise sets PVV_MARGINS_ON in prview.Pv.Disp.Flags.

Sends a PVP_MARGINS message to each of the PRVPAGE objects specified in the prview.pPageView array.

PVV_NEW_PAGE

Move to page

```
VOID pvv_new_page(INT NewPage);
```

Move to page offset NewPage in the print preview range: a page offset of zero corresponds to the first page in the print preview range.

Draws the pages on screen using scrolling whenever possible to ensure speed.

If necessary, rounds NewPage down, or up, to ensure that the pages on screen do not extend beyond the allowed range and then writes the possibly modified value of NewPage to prview.Pv.PageNo.

PVV_INIT

Initialise

```
VOID pvv_init(VOID *xp, INT commid, VOID **ppages);
```

Initialise the PRVVIEW object.

Initialises the PRVVIEW object and starts the print preview operation by sending self a WN_INIT message with as argument the address of an IN_PRVVIEW struct and FALSE: the IN_PRVVIEW struct is set as follows:

calls.hread	specifies the handle of an object that is to be sent read call-back messages by the PAGES active object when it requires the next print element: set to xp.
calls.mread	specifies the read call-back message that is sent by the PAGES active object when it requires the next print element: set to O_LPR_READ.
calls.hdone	specifies the value that is written to prview.hdone: set to self.
calls.mdone	specifies the value that is written to prview.mdone: set to O_PVV_FALSE.
PrintMethod	specifies the value that is written to prview.PrintMethod: set to commid.
Spare1	this is set to zero.
Spare2	this is set to zero.

Writes the handle of the PAGES active object i.e. prview.pPages to *ppages.

Indicates that an `AM_START` message has been sent by writing `TRUE` to `prvview.Started`.

Allows queued active objects to run by sending an `AM_START` message to `w_am`.

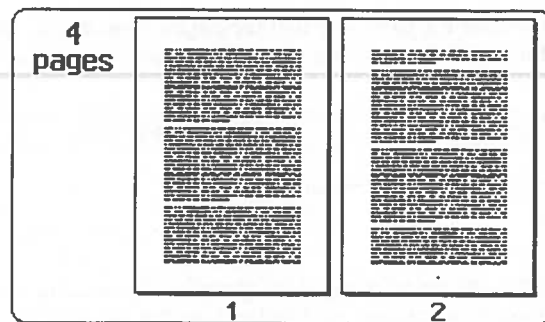
PRVINFO

WIN	PRVINFO
flags id	NoPages FontHeight FontAscent
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_set wn_init

The `PRVINFO` class provides a borderless window containing the page number and either "page" or "pages" as appropriate. An example of such a window is shown in the following picture:

4
pages

The class is normally used as a component in a print preview window as shown in the following picture:



Class definition

Defined in sub-category file `prev.cl` (generated header file `prev.g`).

```

CLASS prvinfo win
{
  REPLACE wn_draw
  REPLACE wn_set
  REPLACE wn_init
  PROPERTY
  {
    INT NoPages;
    UWORD FontHeight;
    UWORD FontAscent;
  }
}

```

Property

<code>prvinfo.NoPages</code>	specifies the number of pages: a textual representation of this value is displayed in the information window.
<code>prvinfo.FontHeight</code>	specifies the height of the boxes in which the page number and associated text is drawn.
<code>prvinfo.FontAscent</code>	specifies the ascent used when drawing text in the boxes.

Resources

Defined in the system resource file.

```
RESOURCE STRING sys_page
{
    str="page";
}

RESOURCE STRING sys_pages
{
    str="pages";
}
```

PRVINFO methods**WN_INIT****Initialise**

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

Initialise property.

Obtains the height and ascent of the font specified by ID `FONT_ID_SWISS_13` and style `G_STY_NORMAL`.

Writes the height of the font to `prvinfo.FontHeight`.

Writes the ascent of the font to `prvinfo.FontAscent`.

WN_SET**Set number of pages**

```
VOID wn_set(INT NoPages);
```

Set the number of pages.

The method simply writes `NoPages` to `prvinfo.NoPages` and then forces a redraw by invalidating the entire content of the window.

WN_DRAW**Draw**

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

Draw the entire display.

Draws a textual representation of the decimal value in `prvinfo.NoPages` in a box in the top left corner of the window.

Draws text in a box immediately beneath the first one. If `prvinfo.NoPages` is one, the text is loaded from the `sys_page` system resource: on English language machines this is "page". Otherwise the text is loaded from the `sys_pages` system resource: on English language machines this is "pages".

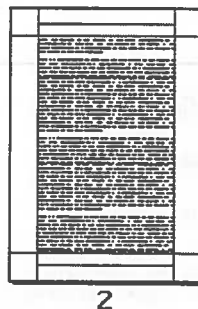
The width and height of each box are `PVV_PAGENO_DISP_GAP` and `prvinfo.FontHeight` respectively.

In both cases the text is centre aligned with an ascent of `prvinfo.FontAscent`.

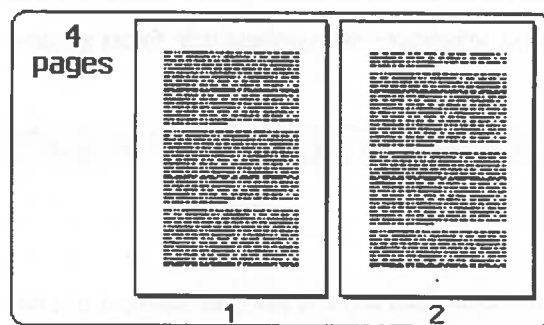
PRVPAGE

WIN	PRVPAGE
flags	init
id	
destroy	wn_dodraw
wn_calc_position	wn_draw
wn_connect	wn_init
wn_dodraw	pvp_margins
wn_emphasise	
wn_key	
wn_position	
wn_redraw	
wn_sense_help	
wn_visible	
wn_set	
wn_sense	
wn_draw	
wn_init	

The PRVPAGE draws an image of a page and the associated page number. An example page view is shown in the following picture:



Notice the margins and the header and footer text. Page views are used as components in the print preview window as shown in the following picture:



In the above picture two PRVPAGE objects were used as components in order to create and maintain the two page images.

In normal use the PRVPAGE object shares the window ID of the main print preview window. The owning object is responsible for supplying the PRVPAGE object with the window ID and its position on screen.

A PRVPAGE object loads the page image from an external memory segment the address of which is read from the property of the parent PRVVIEW object.

Similarly the rectangles that define the layout of the page image are read from the property of the parent PRVVIEW object.

Class definition

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

```

CLASS prvpag win
{
  REPLACE wn_dodraw
  REPLACE wn_draw
  REPLACE wn_init
  ADD pvp_margins
  TYPES
  {
    typedef struct
    {
      PR_PRVVIEW *pPrvView;
      INT Pos;
      UWORD wid;
    } IN_PRVPAGE;
  }
  PROPERTY
  {
    IN_PRVPAGE init;;
  }
}

```

Property

prvpag.init this is an IN_PRVPAGE struct which is passed on initialisation of the PRVPAGE object i.e. by sending it a WN_INIT message.

The IN_PRVPAGE struct is defined as follows:

```

typedef struct
{
  PR_PRVVIEW *pPrvView;
  INT Pos;
  UWORD wid;
} IN_PRVPAGE;

```

the significance of the members of the IN_PRVPAGE struct is as follows:

pPrvView	this is the handle of a parent PRVVIEW object.
Pos	this is an index which determines the offset of the page image in the main window. The offset, in pixels, is equal to the sum of the prvview.Pv.LeftOffset property of the PRVVIEW object and the product of prvpag.Pos and the prvview.Pv.PageViewWidth property of the PRVVIEW object.
wid	this specifies the ID of the main window in which the page view and the page number are drawn.

Resources

Defined in the system resource file.

```

RESOURCE STRING sys_page
{
  str="page";
}

RESOURCE STRING sys_pages
{
  str="pages";
}

```

PRVPAGE methods

WN_DODRAW

Dodraw

VOID wn_dodraw(VOID);

Create an appropriate graphics context and then draw the page view and the page number.

Validates the rectangles enclosing the page image and the page number.

Creates a temporary graphics context.

Draws the page view by sending `self` a `WN_DRAW` message.

Releases the temporary graphics context.

WN_DRAW

Draw

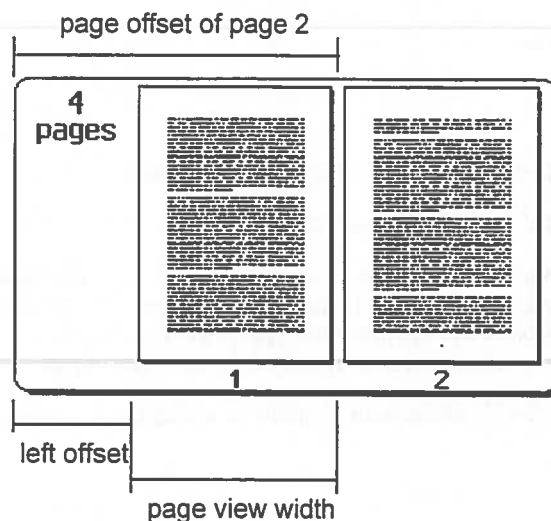
VOID wn_draw(VOID);

Draw on screen the current page view and the associated page number.

If the `prvview.Pv.NoPages` property of the `PRVVIEW` object - i.e. the number of pages to preview - is zero, clears the rectangle enclosing the page view and the page number.

If the current page number is out of the preview range, clears the rectangle enclosing the page view and the page number.

The layout of page views is shown schematically in the following picture.



The quantities of relevance to the `PRVPAGE` class are as follows:

left offset	specified by the <code>prvview.Pv.LeftOffset</code> property of the parent <code>PRVVIEW</code> object.
page view width	specified by the <code>prvview.Pv.PageViewWidth</code> property of the parent <code>PRVVIEW</code> object.
page offset	equal to the sum of the left offset and <code>prvpage.Pos</code> multiplied by the page view width.
page number	equal to the sum of <code>prvpage.init.Pos</code> and the <code>prvview.Pv.PageNo</code> and <code>prvview.Pv.PageOffset</code> property of the parent <code>PRVVIEW</code> object.

Draws the page number in the rectangle specified by the `prvview.Pv.Page.Footer` property of the parent `PRVVIEW` object offset horizontally by the page offset.

Draws the page image in the rectangle specified by the `prvview.Pv.Page.Border` property of the parent `PRVVIEW` object offset horizontally by the page offset.

If the `prvview.Pv.Disp.Flags` property of the `PRVVIEW` object contains `PVV_MARGINS_ON`:

- draws the margins as specified by the `prvview.Pv.Page.Margins` property of the parent `PRVVIEW` object offset by the page offset.

Special note: the page image is loaded from an external memory segment - the handle of this memory segment is read from the `prvview.Pv.PrvSegHandle` property of the parent `PRVVIEW` object.

WN_INIT

Initialise

```
VOID wn_init(IN_PRVPAGE *pinit);
```

Initialise the `PRVPAGE` object.

Writes `*pinit` to `prvpage.init`.

Writes `pinit->wid` to `win.id`.

PVP_MARGINS

Margins

```
VOID pvp_margins(INT flag);
```

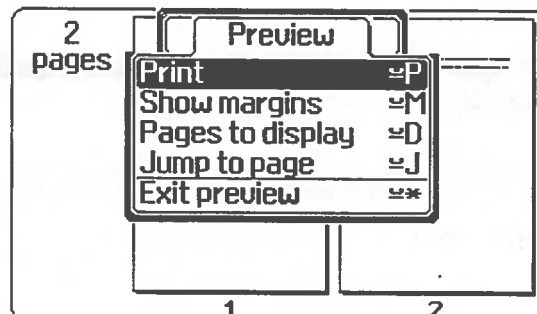
Create a graphics context and the draw the page view and the page number.

Sends `self` a `WN_DODRAW` message.

PRVCOMM

COMMAN	PRVCOMM
	<code>pPrvView</code>
<code>eom_init</code>	<code>com_init</code>
<code>com_statwin</code>	<code>com_menu</code>
<code>com_accl_check</code>	<code>com_file_change</code>
<code>eom_menu</code>	<code>com_exit</code>
<code>com_mode_change</code>	<code>pvc_preview_print</code>
<code>eom_file_change</code>	<code>pvc_preview_margins</code>
<code>eom_exit</code>	<code>pvc_preview_options</code>
	<code>pvc_preview_jump</code>
	<code>pvc_preview_exit</code>

The `PRVCOMM` class provides the print preview command manager. An example print preview menu is shown in the following picture:



The menu items supplied are as follows:

- Print* - this allows the user to print during print preview operations: this option uses the print command in the original command manager of the application.
- Show margins/Hide margins* - this allows the user to simply toggle the display of margins during print preview operations.

- *Pages to display* - this allows the user to set the total number of pages displayed during print preview operations: this is a system wide setting stored in the `PSP` environment variable.
- *Jump to page* - this allows the user to simply set the first page displayed during the print preview operation. This is a local, i.e. not system wide, setting.
- *Exit preview* - this allows the user to terminate the print preview operation thus restoring the original command manager.

Class definition

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

```
CLASS prvcomm comman
{
  REPLACE com_init
  REPLACE com_menu
  REPLACE com_file_change
  REPLACE com_exit
  ADD pvc_preview_print
  ADD pvc_preview_margins
  ADD pvc_preview_options
  ADD pvc_preview_jump
  ADD pvc_preview_exit
  PROPERTY
  {
    PR_PRVVIEW *pPrvView
  }
}
```

Property

`prvcomm.pPrvView` this stores the handle of a `PRVVIEW` object.

Resources

Defined in the system resource file.

```
RESOURCE WSERV_INFO sys_preview_acc
{
  menubar_id=sys_preview_menubar;
  first_com=O_PVC_PREVIEW_PRINT;
  accel=
  {
    'p', /* Print */
    'm', /* Margins */
    'd', /* Pages to display */
    'j', /* Jump to page */
    '*' /* Exit preview */
  };
}
```

PRVCOMM methods

COM_INIT

Initialise

```
VOID com_init(PR_PRVVIEW *pPreView);
```

Initialise the `PRVVIEW` object.

The method simply writes `pPrvView` to `prvcomm.pPrvView`.

COM_MENU

Set menu item text

```
VOID com_menu(WORD menunum,VOID *pArray);
```

Set appropriate text for the *Margins* menu item.

If the `prevview.Pv.Disp.Flags` property of the `PRVVIEW` object contains `PVV_MARGINS_ON`:

- sets the text in the *Margins* menu item from the `SYS_HIDE_MARGINS` system resource: on English language machines this is "Hide margins".

Otherwise:

- sets the text in the *Margins* menu item from the `SYS_DISPLAY_MARGINS` system resource: on English language machines this is "Show margins".

Note: this method is called immediately before the menu is displayed.

COM_EXIT

Exit application

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

Exit the print preview *and* the application.

Destroys the current `PRVVIEW` object by sending a `DESTROY` message to `prvcomm.pPrvView`.

Sends a `COM_EXIT` message to `w_ws->wserv.com`.

PVC_PREVIEW_PRINT

Print

```
VOID pvc_preview_print (WORD menunum, VOID *pArray);
```

Print all of the pages in the print preview range.

If the `prvview.pPages` property of the `PRVVIEW` object specified by `prvcomm.pPrvView` is non-zero:

- displays an information message containing the text in the `SYS_PREVIEW_BUSY` system resource: on English language machines this is "Busy, cannot print yet".

Otherwise:

- it is assumed that the message number of an appropriate print method is specified by the `prvview.PrintMethod` property of the `PRVVIEW` object.
- it is also assumed that the handle of the original command manager is specified by the `prvview.pOldComman` property of the `PRVVIEW` object.
- starts a printing operation by sending the appropriate print message to the original command manager.

PVC_PREVIEW_MARGINS

Toggle margins

```
VOID pvc_preview_margins (VOID);
```

Toggle the visibility of margins in the print preview window.

The method simply sends a `PV_MARGINS` message to the `PRVVIEW` object i.e. `prvcomm.pPrvView`.

PVC_PREVIEW_OPTIONS

Launch *Preview options* dialog

```
VOID pvc_preview_options (VOID);
```

Launch a *Preview options* dialog allowing the user to edit the current print preview display mode. The dialog is passed `prvcomm.pPrvView` as data.

For details of the *Preview options* dialog see the description of the `PRVOPTIONS_DLG` class.

PVC_PREVIEW_JUMP

Launch *Jump to page* dialog

```
VOID pvc_preview_jump (VOID);
```

Launch a *Jump to page* dialog allowing the user to select the number of the current page in the print preview operation. The dialog is passed `prvcomm.pPrvView` as data.

For details of the *Jump to page* dialog see the description of the `PRVJUMP_DLG` class.

PVC_PREVIEW_EXIT**Exit print preview**

```
VOID pvc_preview_exit(VOID);
```

Exit the print preview operation.

Sends a DESTROY message to `prvcomm.pPrvView`.

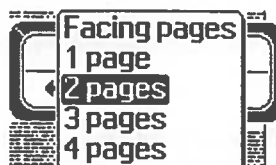
PRVOPTIONS_DLG

WIN	BWIN	DLGCHAIN	DLGBOX		PRVOPTIONS_DLG
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 `PRVOPTIONS_DLG` implements the *Preview options* dialog which allows the user to select the number of pages that are displayed during a print preview operation. An example *Preview options* dialog is shown in the following picture:



The allowed options are shown in the following picture:



The *Facing pages* option allows the display of two pages whereby the first page has an odd page number.

The *Preview options* dialog is used by the `PRVCOMM` print preview command manager described in the present chapter.

Class definition

Defined in sub-category file `prvdlgs.cl` (generated header file `prvdlgs.g`).

```
CLASS prvoptions_dlg dlgbox
{
  REPLACE dl_dyn_init
  REPLACE dl_key
}
```

Property

There is no property associated with the PRVOPTIONS_DLG class.

Resources

Defined in the system resource file.

```

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_options_dialog
{
    title="Display";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED|DLGBOX_NO_DDP;
    controls=
    {
        CONTROL
        {
            class=C_CHLIST;
            prompt="";
            info=CHLIST{rid=sys_preview_options_chlist;};
        }
    };
}

```

PRVOPTIONS_DLG methods

DL_DYN_INIT

Dynamically Initialise

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog: it is assumed that dlgbox.rbuf contains the handle of a PRVVIEW object.

Reads the maximum number of pages from the prview.MaxNoPageViews property of the PRVVIEW object and if the maximum number of pages is less than PVV_VIEW_MAX_PAGEVIEWS:

- deletes the items from the data for the *Mode* control that specify more than the maximum number of pages.
- sets the index of the item selected in the *Mode* control to the smaller of the prview.Pv.Disp.Mode property of the PRVVIEW object and the maximum number of pages.

Otherwise:

- sets the index of the item selected in the *Mode* control to the prview.Pv.Disp.Mode property of the PRVVIEW object.

DL_KEY

Handle key input

```
INT dl_key(VOID);
```

Save the content of the dialog: it is assumed that dlgbox.rbuf contains the handle of a PRVVIEW print preview object.

Sets the number of pages displayed in the print preview by sending a WN_SET message to dlgbox.rbuf with as argument the index of the item selected in the *Mode* control.

Returns WN_KEY_CHANGED.

PRVJUMP_DLG

WIN	BWIN	DLGCHAIN	DLGBOX		PRVOPTIONS_DLG
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 PRVJUMP_DLG class implements the *Jump to page* dialog which allows the user to select the first page on the screen during the print preview operation. An example *Jump to page* dialog is shown in the following picture:



The *Jump to page* dialog is used by the PRVCOMM print preview command manager described in the present chapter.

Class definition

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

```

CLASS prvjump_dlg dlgbox
{
  REPLACE dl_dyn_init
  REPLACE dl_key
}

```

Property

There is no property associated with the PRVJUMP_DLG class.

Resources

Defined in the system resource file.

```
RESOURCE DIALOG sys_preview_jump_dialog
{
    title="Jump to page";
    flags=DLGBOX_NOTIFY_ENTER|DLGBOX_RBUF_FILLED|DLGBOX_NO_DDP;
    controls=
    {
        CONTROL
        {
            class=C_NCEDIT;
            prompt="Page number";
            info=NCEDIT
            {
                low=1;
                high=9999;
            };
        };
    };
}
```

PRVJUMP_DLG methods

DL_DYN_INIT

Dynamically Initialise

```
VOID dl_dyn_init(VOID);
```

Dynamically initialise the content of the dialog: it is assumed that `dlgbox.rbuf` contains the handle of a PRVVIEW object.

Sets the page number of the first page in the preview range as the minimum value in the *Page number* control: this is equal to one plus the `prvview.Pv.PageOffset` property of the PRVVIEW object:

Sets the page number of the first page on screen as the current value in the *Page number* control: this is equal to the sum of the `prvview.Pv.PageOffset` and `prvview.Pv.PageNo` property of the PRVVIEW object

DL_KEY

Handle key input

```
INT dl_key(VOID);
```

Save the content of the dialog: it is assumed that `dlgbox.rbuf` contains the handle of a PRVVIEW object.

If the current value in the *Page number* control exceeds the number of the last page in the preview range:

- sets the current value in the *Page number* control to the number of the last page in the preview range: this is equal to the sum of the `prvview.Pv.PageOffset` and `prvview.Pv.NoPages` property of the PRVVIEW object.
- displays an information message containing text from the `SYS_RANGE_RESET` system resource: on English language machines this is "Out of range - reset to limit".
- returns `WN_KEY_NO_CHANGE`.

Otherwise:

- sets the first page displayed on screen by sending a `PVV_NEW_PAGE` message to `dlgbox.rbuf` with as argument the current value in the *Page number* control minus the page offset i.e. the `prvview.Pv.PageOffset` property of the PRVVIEW object.
- returns `WN_KEY_CHANGED`.



CHAPTER 3

CALENDAR CLASSES

This chapter describes the following two classes:

- the CALIMGWN class which is designed to be used with the CALWIN class.
- the CALWIN class which provides a convenient means to create and use a graphical calendar view.

Note that both the CALIMGWN and CALWIN classes are included in the XADD category and thus an instance of each class should be created as follows:

```
self->demo.calwin=f_new(CAT_MYAPP_XADD,C_CALWIN);  
self->demo.calimgwn=f_new(CAT_MYAPP_XADD,C_CALIMGWN);
```

It is expected that applications that wish to display a calendar window are more likely to create an instance of CALWIN, rather than the more primitive CALIMGWIN.

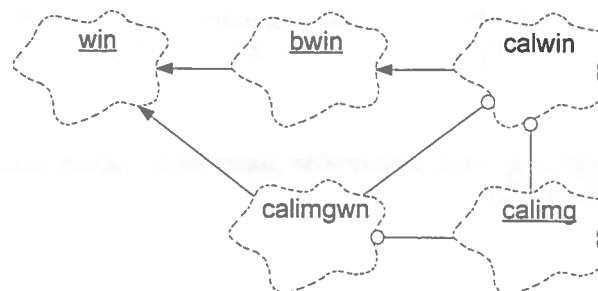
Since CALWIN's component CALIMGWIN expects to receive redraw message, an instance of CALWIN is not suited to being created from OPL or HWIF programs.

Precursors

An understanding of the CALWIN and CALIMGWN classes will be helped by a knowledge of:

- the graphical calendar display in the Series 3a Agenda application.
- the CALIMG class described in *The Calendar Image Class* chapter of the *FORM Reference* manual.
- the WIN and BWIN classes.

Class diagram



CALIMGWN

WIN	CALIMGWN
flags id	calimg
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_redraw

The CALIMGWN class is intended to be used by the CALWIN class described in the next section. It does no more than provide a borderless window suitable for holding a calendar view drawn by an instance of the CALIMG class.

Class definition

The CALIMGWN class subclasses win and is defined in the sub-category file *calwin.cl* (with generated header file *calimgwn.g*).

```

CLASS calimgwn win
{
  REPLACE wn_redraw
  PROPERTY 1
  {
    VOID *calimg;
  }
}

```

Property

calimgwn.calimg an instance of the CALIMG class responsible for drawing and maintaining a calendar view. A description of the CALIMG class may be found in the *FORM Reference* manual.

CALIMGWN methods

WN_REDRAW

Redraw

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

Redraw the part of the calendar view which overlaps the region defined by the P_RECT struct pointed to by prect.

The code is as follows:

```

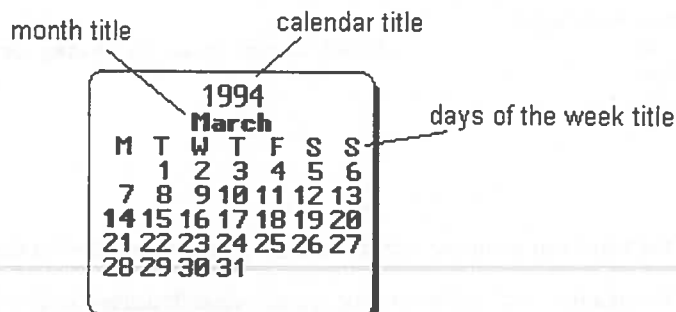
wBeginRedrawGC0(self->win.id,prect);
p_send3(self->calimgwn.calimg,O_CI_REDRAW,prect);
wEndRedraw();

```

CALWIN

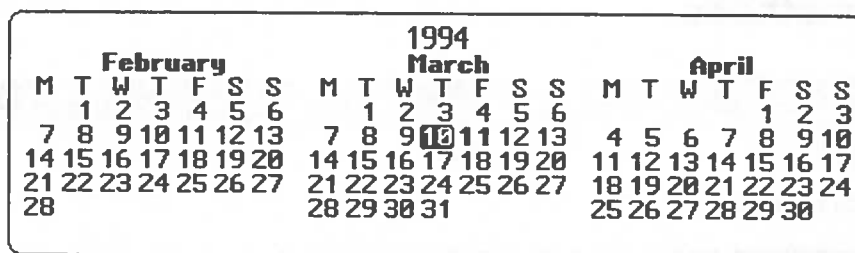
WIN	BWIN	CALWIN
flags id		calimgwn calimg flags self_ptr
destroy wn_calc_position wn_connect wn_dodraw wn_emphasise wn_key wn_position wn_redraw wn_sense_help wn_visible wn_set wn_sense wn_draw wn_init	wn_draw wn_emphasise	destroy wn_init wn_emphasise wn_key wn_sense

The CALWIN class provides a bordered and shadowed window containing a graphical calendar view. An example CALWIN display is shown in the following diagram, which indicates some of the components of the window:



Note that today's date - i.e. the fourteenth - is indicated in a bold font.

On the Series 3a, the calendar also supports multiple rows and columns of months, as illustrated in the following picture:



An owning object may allow the CALWIN class to determine appropriate fonts, font styles and spacings in which case the calendar view may contain either one, three or twelve months. Alternatively the owning object may explicitly specify the fonts, font styles, spacings and the number of rows and columns. Clearly the first mode of use is the simplest and is thus recommended.

Note that, on the *Workabout*, a calendar window may only display a single month.

Class definition

The CALWIN class subclasses bwin and is defined in the sub-category file *calwin.cl* (with generated header file *calwin.g*).

```

CLASS calwin bwin
{
    REPLACE destroy
    REPLACE wn_init           initialise to 1 of 3 types of calendar window
    REPLACE wn_emphasise      deals with cursor drawing & erasing
    REPLACE wn_key
    REPLACE wn_sense
    CONSTANTS
    {
        IN_CALWIN_1_MONTH 0x0001
        IN_CALWIN_3_MONTH 0x0002
        IN_CALWIN_12_MONTH 0x0004
        IN_CALWIN_STD_FLAGS 0x0007      above three Ored together
        IN_CALWIN_USER_SPEC 0x0010     none of the above but user specified
        PR_CALWIN_TODAY_HOOKED 0x0020
    }
    TYPES
    {
        typedef struct
        {
            UWORD flags;
            ULONG days;
            P_POINT pos;
            VOID *calimg;           in_calimg struct (see calimg.cl)
            VOID **self_ptr;        where to write self to (in owner's property)
            } IN_CALWIN;
    }
    PROPERTY 1
    {
        PR_CALIMGWN *calimgwn;
        VOID *calimg;              allows direct call to calimg eg goto_date() etc
        UWORD flags;
        VOID **self_ptr;
    }
}

```

Property

calwin.calimgwn	the handle of an instance of the CALIMGWN class described in the first section of the current chapter.
calwin.calimg	the handle of an instance of the CALIMG class described in <i>The Calendar Image Classes</i> chapter of the <i>FORM Reference</i> manual.
calwin.flags	contains an ored combination of flags as described for the wn_init method.
calimg.self_ptr	assumed to point to the address at which the owning application stores the CALWIN object handle.

CALWIN methods

DESTROY

Destroy

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

Destroy the CALWIN instance.

If calwin.flags contains PR_CALWIN_TODAY_HOOKED, the method sends a WS_HOOK_TODAY_CHANGED message to the WSERV object.

The method then supersends a DESTROY message.

WN_INIT**Initialise**

```
VOID wn_init(IN_CALWIN *init);
```

Initialise the instance of CALWIN according to the content of the IN_CALWIN struct pointed to by init.

If `w_ws->wserv.flags` does not contain `PR_WSERV_FULLSCREEN`, the method calls `p_leave` with an argument of `RUN_ACTIVE_CLEANUP_NONOTIFY`.

Otherwise the method creates, initialises and displays a calendar view.

The appearance of the calendar view is specified by means of an IN_CALIMG struct defined as follows:

```
typedef struct
{
    UWORD flags;
    ULONG days;
    P_POINT pos;
    VOID *calimg;
    VOID **self_ptr;
} IN_CALWIN;
```

The significance of the members of the IN_CALWIN struct is as follows:

flags the allowed values are:

`IN_CALWIN_USER_SPEC` in which case the initialisation data for the CALIMG instance is read from `init->calimg`. Note that this flag takes precedence over the remaining three.

`IN_CALWIN_12_MONTH` in which case appropriate initialisation data is created for a calendar view containing twelve months arranged in two rows of six months each. On the *Workabout*, this flag, if present, will be removed from the initialisation data.

`IN_CALWIN_3_MONTH` in which case appropriate initialisation data is created for a calendar view containing three months arranged as one row of three months. On the *Workabout*, this flag, if present, will be removed from the initialisation data.

`IN_CALWIN_1_MONTH` in which case appropriate initialisation data is created for a calendar view containing one month. On the *Workabout*, this flag will be forced to be present in the initialisation data.

days specifies the initial value for the current date and is expressed as the number of days elapsed since 1/1/1900.

pos specifies the initial position of the main bordered window and is modified if necessary to ensure that the calendar is not clipped by the edges of the screen. In the latter case the modified value is such that the calendar view is centred in the window.

calimg the address of an IN_CALIMG struct containing initialisation data for the CALIMG instance. This is ignored unless the flags member contains `IN_CALWIN_USER_SPEC`. The IN_CALIMG struct is described in *The Calendar Image Class* chapter of the *FORM Reference* manual.

self_ptr specifies an address in the owning application where the handle of the CALWIN instance is stored.

Writes `init->flags` to `calwin.flags` and writes `init->self->ptr` to `calwin.self->ptr`.

On the *Workabout*, the flags `IN_CALWIN_3_MONTH` and `IN_CALWIN_12_MONTH` are cleared from the initialisation data and the flag `IN_CALWIN_1_MONTH` is forced to be set. before the data is copied to `calwin.flags`. Note that this also automatically guarantees that `IN_CALWIN_USER_SPEC` is not present.

Unless `init->flags` contains `IN_CALWIN_USER_SPEC`, an IN_CALIMG struct is created and initialised with appropriate values for the required calendar view. These values are as follows:

wid specifies the window ID of the borderless window: this is set to `calwin.calimgwn->win.id`.

width specifies the width of the borderless window corresponding to the CALIMGWN component. This window is of the exact size required to hold the calendar view. If the calendar view is too wide for the screen the method calls `p_leave` with an argument of `E_GEN_TOOWIDE`.

<code>t1</code>	specifies the gutter dimensions. The <code>x</code> member specifies the width of the left and right gutters and is set to seven. The <code>y</code> member specifies the height of the top and bottom gutters. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , the <code>y</code> member is set to five. Otherwise it is set to seven.
<code>mrow</code>	specifies the number of rows. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , <code>mrow</code> is set to two, otherwise it is set to one.
<code>mcol</code>	specifies the number of columns. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , <code>mcol</code> is set to six, or if <code>init->flags</code> contains <code>IN_CALWIN_3_MONTHS</code> , <code>mcol</code> is set to three, otherwise it is set to one.
<code>flags</code>	set to zero.
<code>title</code>	specifies the font characteristics of the main title. The <code>leading</code> , <code>style</code> and <code>fid</code> members are set to one, <code>G_STY_NORMAL</code> and <code>FONT_ID_13_S</code> respectively (on the <i>Workabout</i> , the title font ID is set to <code>FONT_ID_S3BOLD</code>).
<code>month</code>	specifies the font characteristics of the month title. The <code>leading</code> and <code>style</code> members are set to one and <code>G_STY_NORMAL</code> respectively. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , the <code>fid</code> member is set to <code>FONT_ID_S3BOLD</code> , otherwise it is set to <code>FONT_ID_11B_S</code> (<code>FONT_ID_S3BOLD</code> on the <i>Workabout</i>).
<code>dow</code>	specifies the font characteristics of the days of the week title. The <code>leading</code> and <code>style</code> members are set to one and <code>G_STY_NORMAL</code> respectively. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , the <code>fid</code> member is set to <code>FONT_ID_S3</code> , otherwise it is set to <code>FONT_ID_11_S</code> (<code>FONT_ID_S3</code> on the <i>Workabout</i>).
<code>day</code>	specifies the font characteristics of the day of the month numbers. The <code>leading</code> and <code>style</code> members are set to one and <code>G_STY_NORMAL</code> respectively. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , the <code>fid</code> member is set to <code>FONT_ID_DIGITS_8*4</code> , otherwise it is set to <code>FONT_ID_11_S</code> (<code>FONT_ID_S3</code> on the <i>Workabout</i>).
<code>daygap</code>	specifies a character the width of which in the day number font and style defines the spacing between the day numbers and is set to <code>OSPACE</code> .
<code>mthgapx</code>	specifies the horizontal pixel separation between adjacent months. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , <code>mthgapx</code> is set to six, otherwise it is set to twelve.
<code>hscrlm</code>	specifies the default granularity for scrolling horizontally by month. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , <code>hscrlm</code> is set to six, otherwise it is set to one.
<code>startm</code>	specifies the month number of the first month in the calendar view. If <code>init->flags</code> contains <code>IN_CALWIN_12_MONTHS</code> , <code>startm</code> is set to zero, otherwise it is determined from the value implicitly specified in <code>init->days</code> .
<code>days</code>	specifies the current date expressed as days elapsed since 1/1/1990 and is set to <code>init->days</code> .
<code>font</code>	specifies additional font information for the month, day of week, title and day text. For example, the ascent of the month text, which is <code>font.mth_ascent</code> , is calculated from the font ID and the style specified in <code>month.fid</code> and <code>month.style</code> respectively.

Otherwise (i.e. if `init->flags` contains `IN_CALWIN_USER_SPEC`) it is assumed that `init->calimg` points to an `IN_CALIMG` struct initialised by the owner.

Creates the main bordered window by sending `self` a `WN_CONNECT` message with appropriate arguments. The position is set to the `pos` member of the `IN_CALWIN` struct as described above. The width is set to the width of the calendar plus the width of the left and right gutters, and similarly the height is set to the height of the calendar plus the height of the top and bottom gutters. On the Series 3a, but not on the *Workabout*, if the window is too large to fit on the screen, the method calls `p_leave(E_GEN_TOOWIDE)`.

Creates an instance of the `CALIMGWN` class and writes the handle to `calwin.calimgwn`. Creates the calendar window by sending a `WN_CONNECT` message to `calwin.calimgwn` with a background attribute of `W_WIN_BACK_NONE`. The window dimensions are set to those of the calendar view.

Creates an instance of the `CALIMG` class and writes the handle to both `calwin.calimg` and `calwin.calimgwn->calimgwn.calimg`.

If `init->flags` contains `IN_CALWIN_USER_SPEC`, initialises the `CALIMG` instance by sending a `CI_INIT` message to `calwin.calimg` with `init->calimg` as argument.

Otherwise initialises the `CALIMG` instance by sending a `CI_INIT` message to `calwin.calimg` with the address of an appropriately set `IN_CALIMG` struct as described above.

If `init->flags` contains `IN_CALWIN_12_MONTHS`:

- loads the resource with ID `SYS_CALENDAR` and sets this as the calendar title by sending a `CI_SET_TITLE` message to `calwin.calimg`.

Sends a `WS_HOOK_TODAY_CHANGED` message to `w_ws` with arguments of `calwin.calimg` and `O_CI_TODAY_CHANGED`.

Sets `PR_CALWIN_TODAY_HOOKED` in `calwin.flags`.

Makes the calendar view visible by calling the `hInitVis` utility routine with an argument of `self`.

WN_KEY

Handle keypress

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

Handle a keypress.

If `keycode` is `W_KEY_ESCAPE`:

- returns `WN_KEY_CANCELLED`.

If `keycode` is `W_KEY_RETURN`:

- returns `WN_KEY_CHANGED`.

If `keycode` is `W_KEY_TAB`:

- if `calwin.flags` contains `IN_CALWIN_USER_SPEC`, the method returns `WN_KEY_NO_CHANGE`.
- if `modifiers` contains `W_CTRL_MODIFIER` and `calwin.flags` contains `IN_CALWIN_12_MONTH`, the method returns `WN_KEY_NO_CHANGE`.
- otherwise the method creates a new `CALWIN` object and initialises it as described for the `wn_init` method. If `modifiers` contains `W_CTRL_MODIFIER`, the new calendar contains twelve months. Otherwise if `modifiers` contains `W_SHIFT_MODIFIER`, and `calwin.flags` contains `IN_CALWIN_1_MONTH`, the new calendar contains twelve months. Otherwise if `modifiers` contains `W_SHIFT_MODIFIER`, the new calendar contains half the current number of months. Otherwise if `calwin.flags` contains `IN_CALWIN_12_MONTH`, the new calendar contains one month. Otherwise, the new calendar contains twice the current number of months.
- de-emphasises the current calendar view by sending `self` a `WN_EMPHASISE` message and emphasises the new calendar view by sending a `WN_EMPHASISE` message to the new `CALWIN` object.
- writes the handle of the new `CALWIN` object to the location pointed to by `calwin.self_ptr` and then destroys the current calendar view by sending `self` a `DESTROY` message.
- returns `WN_KEY_NO_CHANGE`.

If `keycode` is less than `0x100` and corresponds to either the special keycode stored at address `&w_ws->wserv.sc[H_SC_ILESS]` or the special keycode stored at address `&w_ws->wserv.sc[H_SC_ICOMMA]` (on English language machines these are the less than symbol and the comma respectively):

- if `modifiers` contains `W_CTRL_MODIFIER`, moves the cursor backwards in time by seven days by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_WEEK`.
- otherwise, moves the cursor backwards in time one day by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_DAY`.

If `keycode` is less than `0x100` and corresponds to either the special keycode stored at address `&w_ws->wserv.sc[H_SC_IMORE]` or the special keycode stored at address `&w_ws->wserv.sc[H_SC_IDOT]` (on English language machines these are the greater than symbol and the full stop respectively):

- if modifiers contains `W_CTRL_MODIFIER`, moves the cursor forwards in time seven days by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_WEEK`.
- otherwise, moves the cursor forwards in time one day by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_DAY`.

If keycode is `W_KEY_UP` and `calwin.flags` contains `IN_CALWIN_1_MONTH`:

- moves the cursor backwards in time one week - i.e. seven days - by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_WEEK` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_RIGHT` and `calwin.flags` contains `IN_CALWIN_1_MONTH`:

- moves the cursor forwards in time one day by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_DAY` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_LEFT` and modifiers contains `W_SHIFT_MODIFIER`:

- moves the cursor backwards in time one day by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_DAY` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_LEFT` and modifiers contains `W_CTRL_MODIFIER`:

- moves the cursor backwards in time one month by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_DAY` and then returns `WN_KEY_NO_CHANGE`.

Otherwise, if keycode is `W_KEY_LEFT`:

- moves the cursor to the left by one day in the calendar view by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_LEFT` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_RIGHT` and modifiers contains `W_SHIFT_MODIFIER`:

- moves the cursor forwards in time one day by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_DAY` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_RIGHT` and modifiers contains `W_CTRL_MODIFIER`:

- moves the cursor forwards in time one month by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_MONTH` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_RIGHT`:

- moves the cursor to the right one day in the calendar view by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_RIGHT` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_UP` and modifiers contains `W_SHIFT_MODIFIER`:

- moves the cursor backwards in time one week by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_WEEK` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_UP` and modifiers contains `W_CTRL_MODIFIER`:

- moves the cursor backwards in time one year by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PREV_YEAR` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_UP`:

- moves the cursor upwards in the calendar view by one day by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_UP`.
- returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_DOWN` and modifiers contains `W_SHIFT_MODIFIER`:

- moves the cursor forwards in time one week - i.e. seven days - by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_WEEK` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_DOWN` and modifiers contains `W_CTRL_MODIFIER`:

- moves the cursor forwards in time one year by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_NEXT_YEAR` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_DOWN`:

- moves the cursor downwards one day in the calendar view by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_DOWN` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_SPACE`:

- moves the cursor to today's date by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_GOTO_TODAY` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_PAGE_UP`:

- moves the cursor to the previous calendar page by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PAGEUP`.

If keycode is `W_KEY_PAGE_DOWN`:

- moves the cursor to the next calendar page by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_PAGEDN`.

If keycode is `W_KEY_HOME`:

- moves the cursor horizontally to the left edge of the calendar view by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_HOME` and then returns `WN_KEY_NO_CHANGE`.

If keycode is `W_KEY_END`:

- moves the cursor horizontally to the right edge of the calendar view by sending a `CI_MOVE_CURSOR` message to `calwin.calimg` with an argument of `CALIMG_END` and then returns `WN_KEY_NO_CHANGE`.

Otherwise the method returns `WN_KEY_NO_CHANGE`.

✶ WN_SENSE

Sense current date

```
VOID wn_sense(ULONG *psense);
```

Write the current date expressed as days elapsed since 1/1/1900 to the `ULONG` pointed to by `psense`.

Sends a `CI_SENSE` message to `calwin.calimg`.

WN_EMPHASISE

Emphasise

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

Emphasise the display if `flag` is `TRUE`, otherwise de-emphasise the display.

Sets the emphasis of the calendar view by sending a `CI_EMPHASISE` message to `calwin.calimg` with an argument of `flag` and then sets the emphasis of the bordered window by supersending a `WN_EMPHASISE` message with an argument of `flag`.

Examples

The following section provides some useful information on using the `CALWIN` class as a component in an application.

Creating a `CALWIN` component

A `CALWIN` object may be created and initialised as follows:

```
IN_CALWIN init;

init.flags=IN_CALWIN_1_MONTH;
p_send(self->kalwin.time,O_TO_SENSE,SENSE_TIME_DAYSEC,&ds);
init.days=ds.day;
init.pos.x=400;
init.pos.y=180;
init.calimg=NULL;
init.self_ptr=&self->kalwin.calwin;
self->kalwin.calwin=f_newsend(CAT_KAL_XADD,C_CALWIN,O_WN_INIT,&init);
p_send(self->kalwin.calwin,O_WN_EMPHASISE,TRUE);
```

The above code creates a calendar view containing one month with the current date set to the date stored in a `TIME` object. Note that large values are specified for the `x` and `y` screen coordinates to force centering of the image.

A description of the `TIME` class may be found in the *OLIB Reference* manual.

Handling keypresses

The following `wn_key` method is suitable for an application that wishes to use a `CALWIN` component. It carries out the following operations:

- if the keypress is a Tab key, and a calendar view is not present, the method launches a calendar window and disables the Menu and accelerator keys by writing `self` to `w_ws->wserv.filter`.
- otherwise the method sends the keypress directly to the calendar view and the return value determines the next action.
- if the return value is `WN_KEY_CANCELLED` the method destroys the `CALWIN` object and re-enables the Menu and accelerator keys by writing `NULL` to `w_ws->wserv.filter`.
- if the return value is `WN_KEY_CHANGED` the method senses the current date is sensed and stores it in the `TIME` object. The method then destroys the `CALWIN` object and re-enables the Menu and accelerator keys by writing `NULL` to `w_ws->wserv.filter`.

```

LOCAL_C VOID DestroyCalwin(PR_KALWIN *self)
{
    w_ws->wserv.filter=NULL;
    hDestroy(self->kalwin.calwin);
    self->kalwin.calwin=NULL;
}

#pragma METHOD_CALL

METHOD INT kalwin_wn_key(PR_KALWIN *self,UINT keycode,UINT modifiers)
{
    IN_CALWIN init;
    P_DAYSEC ds;
    INT ret;

    if (self->kalwin.calwin)
    {
        ret=p_send4(self->kalwin.calwin,O_WN_KEY,keycode,modifiers);
        if (ret==WN_KEY_CANCELLED)
        {
            DestroyCalwin(self);
        }
        else if (ret==WN_KEY_CHANGED)
        {
            ds.sec=0;
            p_send3(self->kalwin.calwin,O_WN_SENSE,&ds.day);
            if ((ret=p_send4(self->kalwin.time,O_TO_SET,SET_TIME_DAYSEC,&ds))<0)
                p_exit(ret);
            DestroyCalwin(self);
        }
    }
    else if (keycode==W_KEY_TAB)
    {
        init.flags=IN_CALWIN_1_MONTH;
        if ((ret=p_send(self->kalwin.time,O_TO_SENSE,SENSE_TIME_DAYSEC,&ds))<0)
            p_exit(ret);
        init.days=ds.day;
        init.pos.x=400;
        init.pos.y=180;
        init.calimg=NULL;
        init.self_ptr=&self->kalwin.calwin;
        self->kalwin.calwin=f_newsend(CAT_KAL_XADD,C_CALWIN,O_WN_INIT,&init);
        p_send(self->kalwin.calwin,O_WN_EMPHASISE,TRUE);
        w_ws->wserv.filter=(PR_WIN *)self;
    }
    return(WN_KEY_CHANGED);
}

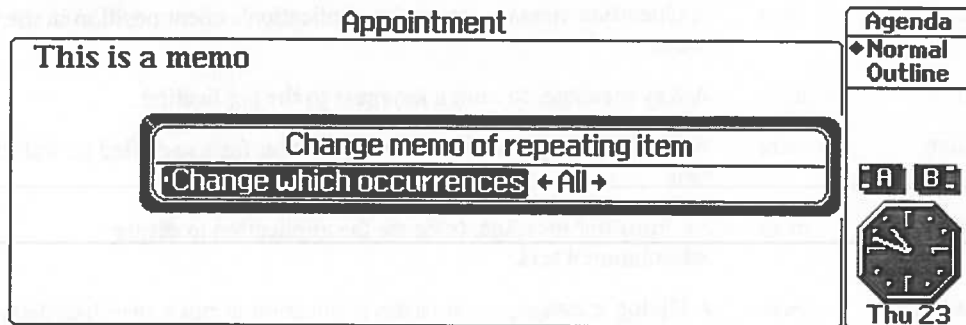
```


CHAPTER 4

AUTOMATIC TEST SYSTEM CLASSES

This chapter documents the `ATSSV` and `ATSTIM` classes that are used to provide Series 3a automatic application test mechanisms, driven from another process.

Although primarily designed for application testing, the mechanism can be used for other purposes. An example is its use by the Series 3a Agenda application, when using the Word application to edit a memo, to force Word to display the dialog shown in the following illustration:



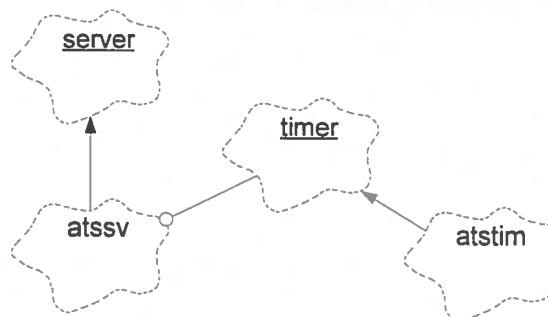
For examples of the use of the ATS mechanism, see the *Series 3a Automatic Test System* chapter of the *Object Oriented Programming Guide*. See also the description of the `ATSDIAL` class in the *Dialog Boxes* chapter of the *HWIM Reference* manual.

Precursors

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

- inter-process messaging, as described in the *Processes and Inter-process Messaging* chapter of the *PLIB Reference* manual
- the `IPCS` and `SERVER` classes, described in the *Inter-process Communication* chapter of the *OLIB Reference* manual
- the `TIMER` class, described in the *Timer Active Object Classes* chapter of the *OLIB Reference* manual.

Class diagram



ATSSV

SERVER	ATSSV
q	timer
t1	pm
t2	freed
cid	pm_key
destroy	sv_init
sv_abrun	sv_run
sv_init	sv_abrun
sv_run	sv_free
	sv_key

The ATSSV class subclasses SERVER to provide an application with the ability to respond to inter-process messages with message types in the range TY_ATS_START_RANGE (0x30) to TY_ATS_END_RANGE (0x3f) inclusive. The following ATS inter-process message types are defined in the header file *ats.h*:

TY_ATS_CLIENT_POS	0x30	A ClientPos message, to set the application's client position in the task order.
TY_ATS_KEY	0x31	A Key message, to send a keypress to the application.
TY_ATS_PAUSE	0x32	A Pause message, to pause the application for a specified period of time.
TY_ATS_MESSAGE	0x33	An InfoPrint message, to cause the application to display informational text.
TY_ATS_DIALOG	0x34	A Dialog message, to cause the application to run a specified dialog.
TY_ATS_SELF_CHECK	0x35	A SelfCheck message, to cause the application to run a self-consistency check.
TY_ATS_WINDOW_XSUM	0x36	A Checksum message, to run a checksum on the application's display.
TY_ATS_RECORD	0x37	A Record message, to start or stop external recording of keypresses received by the application.
TY_ATS_GET_KEY	0x38	A GetKey message, used with Record, to request notification of the next keypress received by the application.
TY_ATS_ALLCOUNT	0x39	An AllocCheck message, to walk the allocated cells in application's heap and report on the results.

Class definition

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

```

CLASS atssv server
{
  REPLACE sv_init
  REPLACE sv_run
  REPLACE sv_abrun
  ADD sv_free
  ADD sv_key
  PROPERTY 1
  {
    PR_TIMER *timer;
    VOID *pm;
    WORD freed;
    VOID *pm_key;
  }
}

```

Property

<code>atssv.timer</code>	Either NULL or the handle of an instance of the <code>ATSTIM</code> timer class, used to implement the pause facility.
<code>atssv.pm</code>	A pointer to an <code>ATS_MESS</code> struct, containing the current ATS message data.
<code>atssv.freed</code>	Set to <code>FALSE</code> on receipt of an inter-process ATS message, and only set to <code>TRUE</code> when the ATS message buffer is freed (by a call to <code>p_mfree</code>).
<code>atssv.pm_key</code>	If not NULL, a pointer to the ATS message data for a Getkey message.

Auxiliary structures

As well as the ATS inter-process message types given earlier in this chapter, the header file *ats.h* contains the declarations of the following structures, used to construct the message buffer for an ATS inter-process message. An application (referred to here as the *controlling* process) can include *ats.h* to allow it to use the ATS mechanism, without having to have any knowledge of the ATS classes themselves (such as would be given by including *atssv.g*).

The `ATS_DIAL_DEF` struct is used to specify a dialog to be run:

```
typedef struct
{
    UWORD main;
    UWORD mainlen;
    UWORD buts;
    WORD butslen;
} ATS_DIAL_DEF;
```

The meanings of the members of this struct are as follows:

<code>main</code>	The offset within the controlling process to a buffer containing the loaded resource for the dialog.
<code>mainlen</code>	The length of the main resource.
<code>buts</code>	Either NULL or the offset within the controlling process to a buffer containing data for one of the dialog's controls. The data may be a choice list resource, a action list resource or a text string (up to 256 characters, including the terminating zero) for an edit box.
<code>butslen</code>	The length of the <code>buts</code> resource.

See the *Series 3a Automatic Test System* chapter of the *Object Oriented Programming Guide* and the description of the `ATSDIAL` class in the *Dialog Boxes* chapter of the *HWIM Reference* manual for further information on the `ATS_DIAL_DEF` struct.

The `ATS_KEY_DEF` struct contains the keycode and the modifier flags for a keypress sent by means of a `TY_ATS_KEY` inter-process message:

```
typedef struct
{
    UWORD key;
    UWORD mod;
} ATS_KEY_DEF;
```

The data of an ATS inter-process message is contained in an `ATS_MESS_BODY` struct:

```
typedef union
{
    UWORD position; /* used for TY_ATS_CLIENT_POS messages */
    ATS_KEY_DEF k; /* used for TY_ATS_KEY messages */
    ATS_DIAL_DEF d; /* used for TY_ATS_DIALOG messages */
    UWORD delay; /* used for TY_ATS_PAUSE messages */
    VOID *offs; /* used for TY_ATS_MESSAGE messages */
    WORD par; /* used for TY_ATS_SELF_CHECK and TY_ATS_RECORD messages */
    UWORD wid; /* used for TY_ATS_WINDOW_XSUM messages */
} ATS_MESS_BODY;
```

and the ATS inter-process message buffer is an `ATS_MESS` struct:

```
typedef struct
{
    E_MESSAGE mess;
    ATS_MESS_BODY u;
} ATS_MESS;
```

See the following description of the `sv_run` method for the usage of the various message buffer members.

When ATS is being used to record keypresses the data for each keypress, in response to a `TY_ATS_GET_KEY` message, is contained in an `ATS_KEY` struct:

```
typedef struct
{
    UWORD time;
    UWORD keycode;
    UBYTE modifiers;
    UBYTE count;
} ATS_KEY;
```

This struct corresponds to the keypress event part of a window server `WS_EVENT_X` struct, defined in *wlib.h*, starting with its `time` member (that is, omitting its leading `handle` member).

ATSSV methods

SV_INIT

Initialise

```
VOID sv_init(VOID);
```

Initialise the ATSSV object.

Supersends the `SV_INIT` message, passing the start and end of the range of acceptable message types as `TY_ATS_START_RANGE` and `TY_ATS_END_RANGE` respectively.

Note that the superclass method sends an `IP_ADD_SERVER` to the instance of (a subclass of) the `OLIB_IPCS` class whose handle is stored in `w_am->appman.ipcs`. This object must therefore have been created before the instance of ATSSV is initialised. On the Series 3a an instance of `IPCS` and an instance of ATSSV are always created and initialised during the initialisation of the `HWIMMAN` application manager.

SV_RUN

Process an ATS message

```
VOID sv_run(ATS_MESS *pm);
```

Process the ATS inter-process message whose message buffer is pointed to by `pm`.

The value of `atssv.freed` is set to `FALSE` and the pointer `pm` is copied to `atssv.pm`. Further processing depends on the inter-process message type, stored in `pm->mess.type`:

TY_ATS_CLIENT_POS

Sets the application's client position by calling:

```
wClientPosition(pm->u.position, 0);
```

and then frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of zero.

TY_ATS_KEY

Processes a keypress received from another process.

The ATS inter-process message is first freed by means of an `SV_FREE` message, with a return value of zero. The method then sends `w_ws` a `WS_PROCESS_KEY` message, with a keycode of `pm->u.key` keycode and a modifiers value of `pm->u.k.mod`.

TY_ATS_PAUSE

Pauses the application.

If `pm->u.delay` is zero the method sends `w_am` an `AM_YIELD` message and then frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of zero.

If `pm->u.delay` is non-zero, `atssv.timer` (an instance of `ATSTIM`) is sent an `AO_QUEUE` message to generate a relative timeout with a time interval of `pm->u.delay` tenths of a second. If `atssv.timer` is `NULL`, an instance of `ATSTIM` is created and initialised, and its handle is written to `atssv.timer`, before the `AO_QUEUE` message is sent. If the creation or initialisation fails, `p_leave` is called, resulting in `ATSSV` receiving an `SV_ABRUN` message. Otherwise, the freeing of the ATS inter-process message is handled by `ATSTIM`, at the completion of the pause.

TY_ATS_MESSAGE

Displays an information message copied from another process.

The message to be displayed is copied from offset `pm->u.off`s in the process with process ID `pm->mess.pid`. This message should be a zero terminated string and may be up to 128 bytes in length, including the terminating zero. Before being displayed in the top left corner of the screen, by means of a call to the window server function `wInfoMsgCorner`, the message is padded with two leading and two trailing spaces.

The method finally frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of zero.

TY_ATS_DIALOG

Runs an ATS dialog.

Creates an instance of `ATSDIAL` (see the *Dialog Boxes* chapter of the *HWIM Reference* manual). If the creation fails, the method frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of `E_GEN_NOMEMORY`.

Otherwise the dialog is sent, under the protection of `p_enter`, a `DL_DYN_INIT` message, passing the pointer `pm` and the handle, `self`, of this instance of `ATSSV`. If the return value from the `DL_DYN_INIT` message is non-zero (indicating that `p_leave` was called) the return value is copied into the dialog's `atsdial.ret` property and the dialog is sent an `O_DESTROY` message.

In all cases where the instance of `ATSDIAL` was successfully created, the freeing of the ATS inter-process message is handled by `ATSDIAL` on destruction of the dialog.

TY_ATS_SELF_CHECK

Performs an application-specific self-consistency check.

Sends `w_ws` a `WS_SELF_CHECK` message, passing `pm->u.par` and then frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of the result returned by the `WS_SELF_CHECK` message.

It is the responsibility of the application's subclass of `WSERV` to provide a meaningful `ws_self_check` method.

TY_ATS_WINDOW_XSUM

Performs a checksum on a specific window or on the whole screen.

Calls the window server function `gInquireChecksum` for the window with ID `pm->u.wid` (an ID of zero performs a checksum on the whole screen).

The method finally frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value containing the result of the checksum calculation.

TY_ATS_RECORD

Starts (if `pm->u.par` is non-zero) or stops (if `pm->u.par` is `FALSE`) keypress recording.

If attempting to start keypress recording when recording is already in progress, the method simply frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of `E_GEN_INUSE`. Otherwise the method sets the keypress recording state (by storing the handle of this instance of `ATSSV` in `datGate->gate.getkeys`) and sends itself an `SV_FREE` message, with a return value of zero.

If stopping keypress recording the keypress recording state is cleared (by clearing `DatGate->gate.getkeys`). If `atssv.pm_key` is non-zero, indicating that a `TY_ATS_GET_KEY` inter-process message is outstanding, that message is freed by a call to `p_mfree`, passing a return value of `E_FILE_CANCEL` and `atssv.pm_key` is set to `NULL`. regardless of the initial value of `atssv.pm_key`, the method finally frees the `TY_ATS_RECORD` inter-process message by sending itself an `SV_FREE` message, with a return value of zero.

TY_ATS_GET_KEY

Transmits the next keypress to another process when in the keypress recording state.

It is a programming error if an inter-process message of this type is received when the application is not in the keypress recording state, as set by the earlier receipt of a `TY_ATS_RECORD` inter-process message.

If `atssv.pm_key` is not `NULL`, indicating that an inter-process message of this type is already waiting to be processed, the method frees the `TY_ATS_RECORD` inter-process message by sending itself an `SV_FREE` message, with a return value of `E_GEN_INUSE`.

Otherwise the method simply copies the message pointer `pm` to `atssv.pm_key`. The message will be freed by the execution of the `sv_key` method, when the application next receives a keypress.

TY_ATS_ALLCOUNT

Checks the application's heap.

Sends `w_ws` a `WS_GET_ALLOC_INFO` message, which walks the application's heap and displays an information message showing the number of allocated cells and the total number of bytes of allocated memory. The method then frees the ATS inter-process message by sending itself an `SV_FREE` message, with a return value of zero.

SV_ABRUN

Process an error

```
VOID sv_abrun(ATS_MESS *pm, INT ret);
```

Provide additional, specific, error processing.

If `atssv.freed` is `FALSE`, indicating that the ATS inter-process message has not yet been freed, the method frees the message by calling:

```
p_mfree(pm, ret);
```

SV_FREE

Free inter-process message

```
VOID sv_free(INT ret);
```

Provide the normal means of freeing an ATS inter-process message after its processing.

Sets `atssv.freed` to `TRUE`, calls:

```
p_mfree(self->atssv.pm, ret);
```

and then sends `w_am->appman.ipcs` an `AO_QUEUE` message to queue a read for the next ATS inter-process message.

SV_KEY

Report a keypress

```
VOID sv_key(ATS_KEY *pkey);
```

Report a keypress to the process that has previously registered an interest by sending an ATS inter-process message of type `TY_ATS_RECORD`. An `SV_KEY` message is received from the `ao_run` method of the application's instance of a subclass of `WSERV`.

If `atssv.pm_key` is `NULL`, the method simply returns. Otherwise, the `ATS_KEY` struct pointed to by `pkey` is copied to the offset `atssv.pm_key->u.offt` in the process with process ID `atssv.pm_key->mess.pid`. Following this, the current ATS inter-process message is freed by calling:

```
p_mfree(self->atssv.pm_key, 0);
```

and `atssv.pm_key` is set to `NULL`.

ATSTIM

ACTIVE	TIMER	ATSTIM
q priority isactive pcb stat		cathand
destroy ao_init ao_cancel ao_abrun ao_queue ao_run	ao_init ao_queue tm_qabsolute	ao_init ao_run

The ATSTIM class is specifically designed for use by ATSSV to implement its Pause function. It is not expected that application code will explicitly create an instance or send messages to any instance.

Class definition

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

```

CLASS atstim timer
{
  REPLACE ao_init
  REPLACE ao_run
  PROPERTY
  {
    VOID *owner;
  }
}

```

Property

atstim.owner The handle of the owning instance of ATSSV.

ATSTIM methods

AO_INIT

Initialise

```
VOID ao_init(VOID *owner);
```

Initialise the ATS timer.

Supersends the AO_INIT message to open a channel to the TIM: device and then copies **owner**, the handle of the owning instance of ATSSV to **atstim.owner**. The final action is to add itself to the application manager's task list, with priority zero, by sending **w_am** an AM_ADD_TASK message.

AO_RUN

Process completion

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

Process completion of the timer interval.

Frees the ATS inter-process message by sending **atstim.owner** an SV_FREE message, with a return value of zero.

CHAPTER 5

ADDITIONAL ACTIVE OBJECT CLASSES

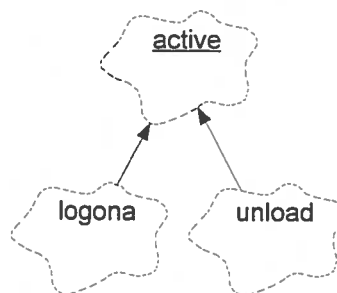
This chapter documents the LOGONA and UNLOAD active object classes.

Precursors

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

- the `ACTIVE` class described in the *ACTIVE Class and Active Objects* chapter of the *OLIB Reference manual*.
- the `p_logona` and `p_logoffa` PLIB routines described in the *Error Handling* chapter of the *PLIB Reference manual*.
- the use of dynamic link libraries: see for example the *Object Oriented Programming* chapter in the *PLIB Reference manual* and the *Building a Dynamic Library* chapter in the *Object Oriented Programming Guide*.

Class diagram



LOGONA

ACTIVE	LOGONA
q priority isactive pcb stat	pid owner mess
destroy ao_init ao_cancel ao_abrun ao_queue ao_run	ao_init ao_cancel ao_queue ao_run

The LOGONA class is used to queue a request for the termination of a process to be reported. The report is implemented by sending a specified message to a specified owning object.

LOGONA adds value to the PLIB `p_logona` routine by packaging the mechanism into an active object, thereby simplifying the detection of the completion of the asynchronous request.

Class definition

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

```

CLASS logona active
{
  REPLACE ao_init
  REPLACE ao_queue
  REPLACE ao_cancel
  REPLACE ao_run
  PROPERTY
  {
    UWORD pid;
    VOID *owner;
    UWORD mess;
  }
}

```

Property

<code>logona.pid</code>	The process ID of the process whose termination is to be reported.
<code>logona.owner</code>	The handle of the object to which a message should be sent on termination of the process.
<code>logona.mess</code>	The method number of the message that is to be sent to <code>logona.owner</code> .

LOGONA methods

AO_INIT

Initialise

```
VOID ao_init(UWORD pid, VOID *owner, UWORD mess);
```

Initialise the LOGONA object.

Initialises property by writing `pid` to `logona.pid`, writing `owner` to `logona.owner` and writing `mess` to `logona.mess`.

Adds `self` to the task queue by sending an `AM_ADD_TASK` message to `w_am`.

Queues a request by sending `self` an `AO_QUEUE` message.

AO_QUEUE

Queue request

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

Queue a request for a report of process termination.

Queues the request by calling the `p_logona` PLIB routine with as arguments `logona.pid` and the address of `active.stat`. The call is made under the protection of the `f_leave` PLIB routine.

AO_CANCEL

Cancel request

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

Cancel a queued request.

If `active.isactive` is non-zero indicating that a request is active:

- cancels the request by calling the `p_logoffa` PLIB routine with an argument of `logona.pid`.
- ensures that the cancel has completed by calling the `p_waitstat` PLIB routine with, as argument, the address of `active.stat`.
- indicates that no request is now outstanding, by writing `FALSE` to `active.stat`.

Note: this method may safely be called if no requests are outstanding.

AO_RUN

Report process termination

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

Report process termination.

Reports process termination as follows:

```
p_send2(logona.owner, logona.mess);
```

Indicates that the event has been consumed by returning `RUN_ACTIVE_USED`.

UNLOAD

ACTIVE	UNLOAD
q priority isactive pcb stat	cathand
destroy ae_init ao_cancel ao_abrun ao_queue ae_run	destroy ao_init ao_run

The `UNLOAD` class is used to queue a request to unload a dynamic library. It provides a convenient means of ensuring that a dynamic library is not unloaded until completion of the current task - or until the next `AM_START` message is sent to the application manager.

An `UNLOAD` object automatically assigns itself a very high priority in order to ensure rapid completion of the request.

The UNLOAD class is intended to be used in a DYL that is loaded by a mechanism such as that provided by the WSERV `ws_launch_dyl` method - that is, where an instance of a single class in the DYL is created and initialised immediately after the DYL is loaded. The UNLOAD class should be made a component of the class in the DYL that is created and initialised and should receive a DESTROY message when that class is destroyed. Since initialisation failures are handled by the mechanism of the `ws_launch_dyl` method, the instance of UNLOAD should not be created and initialised until all other initialisation is complete. Doing this ensures that the instance of UNLOAD will only receive a DESTROY message in circumstances where unloading the DYL is a valid operation.

Class definition

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

```
CLASS unload active
{
  REPLACE destroy
  REPLACE ao_init
  REPLACE ao_run
  PROPERTY
  {
    HANDLE cathand;
  }
}
```

Property

`unload.cathand` The handle of the category to be unloaded.

UNLOAD methods

DESTROY

Destroy

`VOID destroy(VOID);`

Destroy the UNLOAD object.

If `unload.cathand` is non-zero, sends `self` an `AO_QUEUE` message.

Otherwise sends `self` a DESTROY message.

AO_INIT

Initialise

`VOID ao_init(HANDLE cathand);`

Initialise the UNLOAD object.

Writes `cathand`, the handle of the category to be unloaded, to `unload.cathand`.

Assigns itself a high priority by writing `PRIORITY_ACTIVE_POSTER` to `active.priority` and then adds itself to the application manager's task queue by sending `w_am` an `AM_ADD_TASK` message.

AO_RUN

Process completion

`VOID ao_run(VOID);`

Unload the target category.

Unloads the category specified by `unload.cathand` by calling the `p_unloadlib` PLIB routine and then sends `self` a DESTROY message.

The method returns `RUN_ACTIVE_USED`.

INDEX

AO_CANCEL, 5-3
AO_INIT, 4-7, 5-2, 5-4
AO_QUEUE, 5-3
AO_RUN, 4-7, 5-3, 5-4
COM_EXIT, 2-23
COM_INIT, 2-22
COM_MENU, 2-22
DESTROY, 2-3, 2-10, 3-4, 5-4
DL_DYN_INIT, 2-25, 2-27
DL_KEY, 2-25, 2-27
LPR_INIT, 2-3
PVC_PREVIEW_EXIT, 2-24
PVC_PREVIEW_JUMP, 2-23
PVC_PREVIEW_MARGINS, 2-23
PVC_PREVIEW_OPTIONS, 2-23
PVC_PREVIEW_PRINT, 2-23
PVV_DONE, 2-14
PVV_INIT, 2-15
PVV_MARGINS, 2-15
PVV_NEW_PAGE, 2-15
PVV_PAGES_DONE, 2-15
SV_ABRUN, 4-6
SV_FREE, 4-6
SV_INIT, 4-4
SV_KEY, 4-6
SV_RUN, 4-4
WN_DODRAW, 2-20
WN_DRAW, 2-12, 2-17, 2-20
WN_EMPHASISE, 3-9
WN_INIT, 2-12, 2-17, 2-21, 3-5
WN_KEY, 2-11, 3-7
WN_REDRAW, 3-2
WN_SENSE, 3-9
WN_SENSE_HELP, 2-10
WN_SET, 2-12, 2-17

