THE PROFESSIONAL DESKTOP PUBLISHER

Script Language Reference Guide

© David Pilling, P.O. Box 22, Thornton Cleveleys, Blackpool. FY5 1LR.
Web: http://www.pilling.demon.co.uk Email: david@pilling.demon.co.uk

The Program:

Ovation Pro was written by David Pilling.
The new Ovation Pro templates were created by John Ferguson.

The Script Language Reference Guide:

First edition 2000, by David Pilling, Nick Kaijaks, Gavin Crawford and Jan van Vredenburch.
Second edition 2001, by David Pilling, Nick Kaijaks, Gavin Crawford and Jan van Vredenburch.
Third edition 2005, for Windows.

Published by David Pilling.
P.O. Box 22, Thornton Cleveleys, Blackpool. FY5 1LR.
Fax: +41 (0)870 0520941
Email: david@pilling.demon.co.uk
Web: http://www.davidpilling.net

Ovation Pro Script Language

Ovation Pro contains an integrated script language that may be used to control many aspects of its operation, such as the menu organisation, macros and configuration. The script language is an advanced feature of the ***Ovation Pro***, and you do not need to understand it to make full use of the software.

Overview

The integrated script language in ***Ovation Pro*** is based on a simple subset of the C programming language. This chapter outlines the script language, but does not teach you programming. It also describes the library of resident functions that may be called from scripts, and explains how to add your own functions to the library.

If you are familiar with C programming, then the language details given on the following pages should be sufficient to allow you to start using scripts. If you are not familiar with C, then it is recommended that you obtain one of the many beginners books on C that are available.

The important point to note about the script language, is that you do not need to use it or understand it to make full use of ***Ovation Pro***. However, you may wish to use the script language for the following reasons:

- The entire ***Ovation Pro*** menu structure is controlled by the script language. By editing the menu scripts you may re-organise the menus, add menu entries or change the function of menu options. You may also add menus to the button bar.
- The macros system which allows you to define button bar buttons and re-define keys, is based around the script language. An understanding of the script language will allow you to generate your own advanced button and key macros.
- ***Ovation Pro*** choices and preferences are saved in script files. Creating script files of this type allow you to configure the software for different applications.
- Script programs can write text to the caret or read text from the caret, so it is possible to write scripts that generate or modify documents.

Writing & Executing Scripts

Scripts should be plain ASCII text files with the extension .csc or .b24.

You may write scripts using a text editor such as Notepad, or using ***Ovation Pro*** itself. In the latter case you must save the script as ASCII text and not in ***Ovation Pro*** or DDL format *(see 3.10 in the Reference Guide)*.

Executing a Script

A script may be executed by doubling-clicking on it from the desktop, or by dragging it to ***Ovation Pro***.

After execution, the script and all its functions, variables etc. are discarded. Scripts may call resident ***Ovation Pro*** functions, or functions defined in *library scripts*.

Autorun Scripts

If you put a script file in the directory `AutoRun` inside the ***Ovation Pro*** program directory then on running ***Ovation Pro***, it is loaded and its `main()` function is executed.

After execution, the script and all it's functions, variables etc. are discarded. Typically, auto-run scripts are used to configure ***Ovation Pro*** on start-up.

Library Scripts

If you put a script file in the directory `Library` inside the ***Ovation Pro*** program directory then on running ***Ovation Pro***, it is loaded and its `main()` function is executed.

After execution, any functions or global declarations in that file are added to the resident library of functions. Library functions of this type may be called by other scripts, or assigned to a macro and executed from a button or key press.

Multitasking scripts

In order to keep ***Ovation Pro*** calling the Windows GetMessage function you have to place the `pause()` function in the main loop of your program.

Escaping from programs

You can escape from a script at any time by pressing Ctrl Esc.

Error Handling

Any errors in a script are reported in a standard error box.

Example Scripts

A number of example scripts are supplied on the ***Ovation Pro*** release discs. Please refer to the CD guide for details.

Script Language Details

This section gives a brief outline of the script language syntax.

Comments

The characters `/*` introduce a comment, which terminates with the characters `*/`. The C++ form `//` may also be used to introduce a comment which is terminated at the end of the line.

```
/* this is a C style comment
   which may extend across any
   number of lines */
// this is a single line comment
```

Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter or underscore _. Upper and lower case identifiers are different. Identifiers may be any length, and all characters are significant.

Keywords

The following identifiers are reserved for use as keywords:

```
break     else      string
case      for       switch
continue  if        void
default   int       while
do        return
```

Integer constants

An integer constant consisting of a sequence of digits is taken to be decimal. A sequence preceded by `0` (zero) is taken to be octal. Octal constants do not contain the digits 8 or 9. A sequence of digits preceded by `0x` or `0X` (zero, X) is taken to be a hexadecimal integer. The hexadecimal digits include `a` or `A` through to `f` or `F` representing values 10 through to 15.

For example, decimal 127 can be written as follows:

```
a = 127;  // decimal constant
b = 0177; // octal constant
c = 0x7f; // hexadecimal constant
d = 0X7F; // hexadecimal constant
```

Character constants

A character constant is a character enclosed in single quotes, as in `'x'`. The value of the character constant is the ASCII value of the character. In addition, the following escape sequences may be used to represent special characters.

```
carriage return    CR    ASCII 13  \r
newline            LF    ASCII 10  \n
horizontal tab     HT    ASCII 9   \t
double quote       "     ASCII 34  \"
back slash         \     ASCII 92  \\
```

String constants

A string constant is a sequence of characters surrounded by double quotes. A null byte is appended to strings so that programs that scan the string can find the end.

```
s = "ABC";        // set s to ABC
```

In order to represent string constants the character constant escape sequences listed above may be used.

```
s = "Hello\r";    // set s to HelloCR
s = "Hi\n";       // set s to HiLF
s = "*\t*";       // set s to *HT*
s = "\"DEF\"";    // set s to "DEF"
s = "\\";         // set s to \
```

When the string is to be processed by ***Ovation Pro***, for example by the type() function or as a macro body, non printable characters can also be introduced using escape sequences commencing with `|` *(see Appendix B in the main reference manual)*. Use `||` to represent `|`.

```
s = macro("|!=");     // set s to ½ that is |! 128 + the ASCII value of '='
s = macro("||2||");   // set s to |2|
```

Strings may expand macros enclosed in braces `{}`. Use `{{` to represent `{`.

```
s = macro("{MAC}");   // set s to MAC
s = macro("{{3}");    // set s to {3}
```

Strings may expand system variables enclosed in angle brackets `<>`. Use `<<` to represent `<`.

```
p = macro("<%USERNAME%>");    // set p to environment variable
q = macro("<<4>");            // set q to <4>
```

Variable declarations

Variables may be Global or Automatic. Global variables are declared outside of functions and retain their value throughout the program. Automatic variables may be function parameters or declared inside compound statements, and only exist whilst execution is taking place within their scope.

Global and automatic variables may have the same identifiers.

Three types of variables are supported:

- void
- int (32-bit signed integers)
- string (variable length character strings).

```
int nul = 0;      // global declaration
void main(void)
{
   int a, b, c;   // automatic
   int x44 = 99;  // declarations
   string s;
   string str = "Ovation Pro";
   int d = 2;
}
```

Function declarations

Functions can be of type `void`, `int` or `string`, and arguments may be `int` or `string`. Parameters are normally 'call by value', which means that the called function cannot directly alter a variable in the calling function; it can only alter its private, temporary copy. If an ampersand & is placed before a string parameter in a function definition, 'call by reference' is used. In this case the address of the string variable is passed to the called function, which may modify the variable in the calling function. 'Call by reference' may only be used with string parameters.

Like normal C programs, each program must have a function `main()` which is executed first. This implementation does not support function prototypes, so function definitions must appear before they are called in the source file.

```
int length(string s)

{
  int i;
  for (i = 0; schar(s, i) != 0; i++);
  return i;
}

void main(void)
{
  type(stoi(length("hello")));
}
```

In the example above, the value that `length` computes is returned to `main()` by the `return` statement. Note that in real programs the built in function `slen()` would be used to find the length of a string.

Operator Precedence

The table below gives the order of precedence and associativity of functions.

Operators	Associativity
()	left to right
! ~ ++ -- + - &	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right

The unary - and + operator has higher precedence than the binary form.

List of Operators

The operators available are similar to those in C, but in this implementation many operators may also be applied to strings.

`l + r`
Addition operator. If `l` and `r` are ints, returns the arithmetic sum.

If `l` and `r` are strings, returns result of concatenating `r` to `l`.

`l - r`
Subtraction operator. `l` and `r` must be ints, in which case returns `r` subtracted from `l`.

`l * r`
Multiplication operator. If `l` and `r` are ints, returns the product.

If `l` is a string and `r` an int, returns `l` concatenated to itself, `r` times.

If `r` is a string and `l` an int, returns `r` concatenated to itself, `l` times.

`l / r`
Division operator. If `l` and `r` are ints, returns the quotient.

If `l` and `r` are strings, returns position of `r` in `l`.

`l % r`
Modulus operator. If `l` and `r` are ints, returns the remainder.

If `l` and `r` are strings, returns `strspn(l, r)`. The position of the first character in `l` not in `r`.

`l & r`
Bitwise AND operator. If `l` and `r` are ints, returns bitwise AND.

`l | r`
Bitwise OR operator. If `l` and `r` are ints, returns bitwise OR.

`l ^ r`
Bitwise XOR operator. If `l` and `r` are ints, returns bitwise XOR.

`l << r`
Left shift operator. If `l` and `r` are ints, returns `l` shifted left `r` bits.

If `l` is a string and `r` an int, returns `l` shifted left `r` characters.

`l >> r`
Right shift operator. If `l` and `r` are ints, returns `l` shifted right `r` bits.

If `l` is a string and `r` an int, returns `l` shifted right `r` characters.

`+l`
Upper case operator. If `l` is a string, forces it to upper case.

`-l`
Unary minus operator. If `l` is an int, does Unary `-`.

If `l` is a string, forces it to lower case.

`~l`
One's complement operator. If `l` is an int, does One's Complement.

If `l` is a string, swops case.

`!l`
Logical NOT operator. `l` must be an int, in which case TRUE (!=0) becomes FALSE (0), and vice versa.

`l && r`
Logical AND operator. `l` and `r` must be ints. Expression evaluates left to right, and terminates as soon as falsehood is known.

`l || r`
Logical OR operator. `l` and `r` must be ints. Expression evaluates left to right, and terminates as soon as truth is known.

`l , r`
Comma operator. `l` is evaluated then `r`.

`l ? r : r1`
Conditional expression. If `l` is TRUE, returns `r` else returns `r1`.

`l == r`
Equality operator. `l` and `r` must be either ints or strings.

`l != r`
Inequality operator. `l` and `r` must be either ints or strings.

`l > r`
Greater than operator. `l` and `r` must be either ints or strings.

`l < r`
Less than operator. `l` and `r` must be either ints or strings.

`l >= r`
Greater or equal operator. `l` and `r` must be either ints or strings.

`l <= r`
Less or equal operator. `l` and `r` must be either ints or strings.

`++l, l++`
Increment operators. `l` must be an int, and an lvalue.

`--l, l--`
Decrement operator. `l` must be an int, and an lvalue.

`l = r`
Assignment operator. `l` and `r` must be of the same type. `l` must be an lvalue.

`l += r`
Assign `l + r` to `l`. `l` and `r` must be of the same type. `l` must be an lvalue.

`l -= r`
Assign to l. `l` and `r` must be ints. `l` must be an lvalue.

`l *= r`
Assign `l * r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

l /= r
Assign `l / r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

`l %= r`
Assign `l % r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

`l |= r`
Assign `l | r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

`l &= r`
Assign `l & r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

`l ^ r`
Assign `l ^ r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

`l <<= r`
Assign `l << r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

`l >>= r`
Assign `l >>r` to `l`. `l` and `r` must be ints. `l` must be an lvalue.

Statements

In a `while` loop, the statement is executed repeatedly as long as the value of the expression remains true (!= 0). The test occurs before execution of the statement.

`while` (*expression*) *statement*;

In a `do` loop, the sub-statement is executed repeatedly so long as the value of the expression remains true (!= 0). The test occurs following each iteration.

`do` *statement* `while` (*expression*)

In a `for` loop the first expression is evaluated once, and thus initialises the loop. The second expression is evaluated before each iteration, and if it becomes false (0) the loop terminates. The third expression is evaluated after each iteration.

`for` (*expr1 ; expr2 ; expr3*) *statement*;

In the `if` statement the expression is evaluated, and if it is true (!= 0), the first statement is executed, otherwise the second statement is executed. An `else` is always connected with the last else-less if at the same block level nesting.

`if` (*expression*) *statement*; `else` *statement*;

The `switch` statement causes control to be transferred to the `case` statement which matches the value of the expression. There may be one `default` statement to which control is transferred if no case constant matches the expression.

```
switch ( expression )
{
 case int constant : statement;
                     break;
 default:            statement;
                     break;
}
```

The `break` statement causes an immediate exit from a loop.

The `continue` statement causes control to pass to the start of a loop.

The Resident Function Library

The remainder of this document describes the resident functions that may be accessed from the script language. These functions are split into two groups, and are summarised below.

System Functions

These functions are mainly concerned with accessing Windows.

- Windows functions
- String functions
- File input/output functions

Ovation Pro Functions

These functions carry out operations in ***Ovation Pro***.

- Macro functions
- Menu control functions
- Applet and Help menu functions
- Choices functions
- File menu functions
- Edit menu functions
- View menu functions
- Text/Picture menu functions
- Style menu functions
- Font Functions
- Object menu functions
- Page menu functions
- Misc menu functions
- Picture functions
- Colour functions
- Document tagged Data Manager
- Printing Functions
- Imposition functions
- Imposition Support functions
- Window functions
- Bookmark functions
- Miscellaneous functions
- Colour Separations Applet functions
- Event Handling functions
- Constant Macros

System Functions

Windows Functions

clock()

```
int clock(void);
```

Returns the time in centi-seconds since ***Ovation Pro*** was run.

messagebox()

```
int messagebox(string &s);
```

Displays a message box containing the message `s` and a **Continue** icon. Returns 1.

confirm()

```
int confirm(string &s);
```

Displays a confirmation box containing the message `s` and **Yes/No** icons. Returns 1 for **Yes**, 0 for **No** or -1 for Esc (or if the user clicks on the background).

confirmparent()

```
int confirmparent(string &s,int parent);
```

As `confirm()` but displays the confirmation box as a child window of parent.

errorbox()

```
void errorbox(string &s);
```

Raises an error, displays a standard error box containing the message `s`, and terminates the script.

exit()

```
void exit(int status);
```

Terminates script execution. The argument `status` is included for compatibility with C.

getenvs()

```
int getenvs(string &s);
```

Gets the value of the system variable specified by `s`. If the variable exists, returns 1 with the value in `s`, otherwise returns 0.

startscan()

```
void startscan(void);
```

Initialises the function `nextobject()`.

nextobject()

```
int nextobject(string &dir, string &wildcard, string &name);
```

Searches directory `dir` for the next object which matches the wildcarded name `wildcard`, and returns its name in the string name. Returns 1 if the object is found, 2 if it is a directory or 0 if there are no more objects. This function should be initialised once before use, using `startscan()`.

objectexists()

```
int objectexists(string &name);
```

Returns 1 if `name` is a file, 2 if it is a directory or 0 if it does not exist.

oscommand()

```
oscommand(int parent,string & verb,string & file,string &
params,string & directory,int flags,string & result);
```

Performs a ShellExecute() (see the Windows documentation). Returns 1 if the is an error with the error message in `result`, or returns 0 with `result` set to "" if there is none. The `flags` parameter is Ovation Pro specific and no values are currently defined. The rest of the parameters are as for ShellExecute().

osversion()

```
int osversion(void);
```

Returns (GetVersion()&(~0x7FFF0000)|0x40000000) - See the Windows documentation for GetVersion(). In other words the top bit is set for Windows 95,98 and ME and is clear for NT, 2000, XP etc. The next to top bit is set on Windows and clear on RISC OS.

osrunexe()

```
int osrunexe(string & commandline);
```

Runs the exe which starts the command line using the rest of the command line for parameters. This function does not return until the exe process has terminated (unlike `oscommand()`), it returns the exit code from the process. I invented this function because `systems()` seemed to have difficulties if the command line had any file names with spaces in.

systems()

```
int systems(string &s);
```

Passes string `s` to the command line interpreter for execution as a command. If `s` is prefixed by the string `call:`, the calling application is copied to the top of memory first. When the called application terminates, the calling application is restored. Returns 0 if successful, otherwise returns -2.

printi()
```
void printi(int i);
```
Prints integer `i` to the vdu.

prints()
```
void prints(string &s);
```
Prints string `s` to the vdu.

debprinti()
```
void debprinti(int value);
```
Print an integer to the debug window.

debprints()
```
void debprints(string & value);
```
Print a string to the debug window.

pause()
```
void pause(int time);
```
Causes a delay of `time` centi-seconds. WIMP polling will continue in this function, so it should be used in the main loop of your program if you want the rest of the programs on the desktop to multitask.

keypress()
```
void keypress(int key);
```
Immitates the character `key` being typed. This is not limited to typing into documents it also applies when any Windows control owned by Ovation Pro has the focus.

isalt()
```
int isalt(void);
```
Returns true if the Alt key is pressed.

isctrl()
```
int isctrl(void);
```
Returns true if the Control key is pressed.

isshift()

```
int isshift(void);
```

Returns true if the Shift key is pressed.

beep()

```
void beep(int beeptype,int frequency,int duration);
```

Produces a beep noise, for values of the parameters greater than zero the Windows API Beep() function is used and so this function has the same limitations. For `frequency` value zero, the `duration` parameter is passed to the API MessageBeep() function. A zero value for both `frequency` and `duration` produces MessageBeep(MB_OK).

String Functions

chars()
```
string chars(int c);
```
Returns a string consisting of the character equivalent of the ASCII code `c`.

itos()
```
string itos(int i);
```
Returns the integer `i` converted to a string.

itoxs()
```
string itoxs(int i);
```
Returns the integer `i` converted to a hexadecimal string.

itors()
```
string itors(int number);
```
Returns a string representing 'number' in upper case Roman numeral format. It is easy to use the unary `'-'` operator to convert this to lower case format.

mids()
```
string mids(string &string, int from, int len);
```
Returns a segment of string `s`, `len` characters in length commencing at character `from`.

schar()
```
int schar(string &s, int n);
```
Returns the ASCII code for character `n` of string `s`.

slen()
```
int slen(string &s);
```
Returns the length of string `s`.

stoi()
```
int stoi(string &s);
```
Returns the string `s` converted to an integer.

sins()

```
int sins(string &search, string &target, int from);
```

Finds the position of `target` in `search`, starting from position `from`. Returns -1 if the `target` is not found.

findid()

```
int findid(string & name);
```

Finds `name` in the script language symbol table. The return value consists of three bytes.

The lowest is the type:
0 void
1 integer
2 string

The next is the mode:
0 global
1 stack
2 reference
3 function
4 internal function

The final byte is the argument count for a function.

Using this function you can discover if variables and functions exist.

If the name does not exist a value of -1 is returned.

File Input/Output Functions

fileopen()

```
int fileopen(string &name, string &mode);
```

Opens the file `name` and returns a handle associated with the file, otherwise returns 0 if the operation fails. The string `mode` should be one of the following sequences:

r	open text file for reading
w	create text file for writing
a	append; open or create text file for writing at end of file
r+	open text file for update i.e. reading and writing
w+	create text file for update
a+	append; open or create text file for update, writing at end

If `mode` includes a `b` after the initial letter (e.g. `rb`), it indicates a binary file instead of a text file.

fileclose()

```
int fileclose(int handle);
```

Closes the file `handle`. Return non-zero if the operation fails.

filegetc()

```
int filegetc(int handle);
```

Returns the next character from the file `handle` (converted to an int), or -1 if an error occurs.

fileputc()

```
int fileputc(int char, int handle);
```

Writes character `c` (converted to an `int`) to file `handle`. Returns the character written, otherwise return -1 if an error occurs.

filereadi()

```
int filereadi(int handle);
```

Returns the next `int` from the file `handle`, or -1 if an error occurs.

filewritei()

```
int filewritei(int i, int handle);
```

Writes integer `i` to file `handle`, or returns non-zero if an error occurs.

filereads()

```
int filereads(string &s, int handle);
```

Reads the next string terminated by a Newline from the file `handle` into the string `s`. The Newline character is included in the string, which is terminated by a null character. Returns non-zero if an error occurs.

filereadtext()

```
int filereadtext(string &s, int handle);
```

Reads the next string terminated by an end of line sequence from the file `handle` into the string `s`. The end of line characters are not included in the string, which is terminated by a null character. Returns non-zero if an error occurs.

filewrites()

```
int filewrites(string &s, int handle);
```

Writes string `s` to file `handle`, or returns non-zero if an error occurs.

fileeof()

```
int fileeof(int handle);
```

Returns non-zero if the end-of-file indicator for the file `handle`, is set.

fileerror()

```
int fileerror(int handle);
```

Returns non-zero if the error indicator for the file `handle`, is set.

fileseek()

```
int fileseek(int offset, int handle);
```

Sets the file position for the file `handle` to the position `offset` characters from the start of the file. Returns non-zero on an error.

filetell()

```
int filetell(int handle);
```

Returns the current file position for the file `handle`, or -1 on an error.

The following three functions allow access to scrap or temporary files. The idea is that a handle to a scrap file name is obtained. This handle continues to exist until disposed of and is for the exclusive use of its creator. A function is provided to get the file name from the scrap handle.

The usual sequence of events will be:

Create scrap handle

Obtain file name

Create file

Perform file system operations

Delete file

Remove scrap handle

scrapcreate()

```
int scrapcreate(void);
```

Return a scrap file handle.

scrapname()

```
int scrapname(int scraphandle,string & scrapfilename);
```

Obtain scrap file name from scrap file handle.

scrapremove()

```
void scrapremove(int scraphandle);
```

Destroy the scrap file handle.

Ovation Pro Functions

Macro Functions

Macros can return values implicitly by setting the variable `'macv'`.

delallmacros()

```
void delallmacros(void);
```

Deletes all macros.

delmacro()

```
void delmacro(string &name);
```

Deletes the macro `name`.

defmacro()

```
void defmacro(int type, string &name, string &defn, string &button);
```

Defines the macro called `name` with the definition specified by `defn`. The macro type is determined by the parameter `type`:

Value	Type of macro
0	The macro is a Button macro
1	The macro is a Key macro
2	The macro is a User macro

The button bar behaviour is changed by adding the following values to `type`:

4	The button is displayed on the button bar
8	A gap is inserted before the button
16	The button is the first in a new row
32	The macro is external and will not be saved with the program's macros
64	The button supports extended events
128	The button supports extended text
0x1FF00	Text button width mask

The parameter `button` is the name of the sprite used to represent the button on the button bar. If `button` is a null string, the macro name will be used as text for the button instead of a sprite.

If the extended event flag is set, then the macro will be expected to respond to extended events. Traditionally a macro is just executed and returns a string. By setting this flag other entry conditions are enabled. These include different sorts of clicks on buttons, e.g. a double click or a click with different mouse buttons. The button is also expected to return either the text for a text button or the sprite name for a graphic button.

If the extended text flag is set the button is a text button and the text button width, 9 bit wide field should be used to set its maximum width. The point is that it is not desirable for the button bar to have to be redrawn every time the text in a button is changed.

When in a macro it is possible to access integer variables `'macenv'` and `'macview'`. The latter is the handle for the view the macro is currently attached to.

The variable `'macenv'` is split into fields.

bits 0..7 where the macro is called from

2 - key press

1 - button bar

0 - anything else

bits 8..15 why the macro is being called

for a button

1 click with select

2 double click with select [not implemented]

3 drag with select

4 click with adjust

5 double click with adjust [not implemented]

6 drag with adjust

7 click with menu

8 return button text/sprite name

bits 16..23 extra data

for a button click

bit 16 shift key state

bit 17 control key state

bit 18 alt key state

bits 24..31 calling depth

increases by one each time a macro calls a macro

(which means you can work out if a macro is called at the top level)

This is an example macro which shows the value of the system clock and enters a string in the text when clicked on;

```
{if((macenv & 0xFF00)==0x800) macv=itos(clock()); else
{type(itoxs(macview));refreshbutton("xxxxxxxx",macview,0);}}
```

defmacro2()

```
void defmacro2(int type, string &name, string &defn, string &button,
string &helpstr);
```

Define a macro. Syntax the same as `defmacro()`, but with additional parameter `helpstr` for providing help on buttons.

defmacro3()

```
void defmacro3(int type, string &name, string &defn, string &button,
string &helpstr,string & aname);
```

Define a macro. Syntax the same as `defmacro2()`, but with additional parameter `aname` for an applet

name. If the applet is not loaded the macro won't be. This provides a way of binding macros to applets. A long running problem has been orphaned macros, an applet is installed and defines a macro, the user saves their macros, subsequently they remove the applet, and the macro remains. The applet name is the same one that appears in the applet Info file.

defmacro4()

```
void defmacro4(int type, string &name, string &defn, string &button,
string &helpstr,string & aname,string & tooltip);
```

Define a macro. Syntax the same as `defmacro2()`, but with additional parameter `tooltip` for a tooltip. If no tooltip is provided the tooltip is synthesised from the help string, as the string up to the first line break or period followed by space or line break.

isdefmacro()

```
int isdefmacro(string &name);
```

If the macro with the specified name exists, return the `type` value from `defmacro()`, otherwise return -1.

macro()

```
string macro(string &s);
```

Evaluates a macro and returns the value.

refreshbutton()

```
void refreshbutton(string & name,int view,int file);
```

If view is null then all button bars for the file will be refreshed, if view is null and file is null then all button bars in the program will be refreshed.

macrorename()

```
void macrorename(string & oldname,string & newname);
```

Renames a macro.

macrostring()

```
int macrostring(string & name,string & macrostring,int n);
```

Allows the various strings associated with macros to be read. Returns the type of the macro. The string read depends on the value of `n`:

```
n==0 button
n==1 defn
n==2 helpstr
n==3 aname
```

compare with `defmacro()`, `defmacro2()`, `defmacro3()`.

Menu Control Functions

create_menu()

```
int create_menu(string &name);
```

Creates a new menu called `name`, and returns a handle associated with the menu.

delete_menu()

```
void delete_menu(int handle);
```

Deletes the menu identified by `handle`.

rementry_menu()

```
void rementry_menu(int handle, int entry);
```

Removes the entry identified by `entry` from the menu identified by handle.

open_menu()

```
void open_menu(int handle);
```

Displays the menu identified by `handle`.

string_menu()

```
string string_menu(int handle, int entry);
```

Returns the text string for the entry identified by `entry` from the menu identified by `handle`.

content_name()

```
string content_name(void);
```

This function is used by ***Ovation Pro*** to toggle an option on the main menu between **Text** and **Picture**, depending upon the active object type.

content_menu()

```
int content_menu(int entry);
```

This function is used by ***Ovation Pro*** to open the correct menu when the user clicks on **Text** or **Picture** on the main menu.

content_menuname()

```
string content_menuname(void);
```

Returns the name of the selected content menu for use in the menu bar i.e. Text or Picture.

addentry_menu()

`int addentry_menu(int handle, string &entry, string &flags, string &submenu, string &key, string &name);`
Add an entry called `name` to the menu identified by `handle`.

The parameter `flags` specifies the name of a function that is executed when the menu is opened. The number of the entry chosen is passed as an `int` to the function. The name of the entry is passed as a `string` to the function, and can be changed in the function if required.

Typically, the function should be of the form:

```
int main_flags(int n, string &s)
{
  int bits;

  switch(n)
  {
    case 0:
          // change entry 1
          if(...) text = "Ungroup"
          break;
    case 1:
          // dotted line after
          bits |= DOTTED;
          break;
  }
  return(bits);
}
```

Bits set in the return value control the flags for the menu entry:

Bit	Macro name	Function when set
0	TICKED	Tick entry
1	DOTTED	Dotted line below entry
4	SHADED	Shade entry

The parameter `entry` specifies the name of the function that is executed when an entry on the menu is chosen. The number of the entry chosen is passed as an `int` to the function. Typically, the function should be of the form:

```
int main_entry(int n, int subcode)
{
  switch(n)
  {
    case 0:
          // deal with entry 1
          break;
    case 1:
          // deal with entry 2
          break;
  }
return subcode;
}
```

If `submenu` is a null string, the menu entry will not have a sub-menu, otherwise `submenu` is the name of a function that returns the handle of the sub-menu. Typically, the function should be of the form given below, but it may actually create or alter the menu if required.

```
int main_menu(int state)
{
  return main_handle;
}
```

If `key` is a null string, the entry will not have a keyboard short-cut, otherwise `key` specifies the keyboard short-cut for the entry. Key names are the same as those used for macro definitions e.g. `C_V` for Ctrl V *(see 17.5 & Appendix C in the Reference Guide)*.

addentryl_menu()

```
int addentryl_menu(int handle, string &entry, string &flags, string
&submenu, string &key, string &name);
```

As `addentry_menu()` except that it takes a literal string for the parameter name i.e. any tokens in `name` are not translated.

insertentry_menu()

```
insertentry_menu(int handle, int position, string &entry, string
&flags, string &submenu, string &key, string &name);
```

As `addentry_menu()` except for the additional second parameter `position` which specifies the position at which the entry is to be added. So if you want to insert an entry before entry 2, `position` should be set to 2.

Parameter `name` may contain translatable text containing tokens in the form {TOK}. The string is translated by having any tokens replaced by the corresponding text from the Messages file.

menudefstring()

```
int menudefstring(int handle, int entry, int index, string &s);
```

When a menu entry is defined, 5 strings are supplied; `entry`, `flags`, `submenu`, `key` and `name`. This function returns in `s` any one of these strings for the menu and entry specified.

The parameter `index` specifies which string is returned:

Index	Meaning
0	Return entry string
1	Return flags string
2	Return submenu string
3	Return key string
4	Return name string

This function returns 0 if `entry` is greater than the number of entries on the menu.

menuprevious()

```
int menuprevious(int n);
```

Within a menu open, flags or entry function, this returns the index of the `n`'th parent menu entry or -1 if there isn't one.

writemenustring()

```
void writemenustring(int handle,int index,string & text);
```

Write a menu item.

Applet and Help Menu Functions

openinfobox()

```
void openinfobox(void);
```

Open program information window.

openappletbox()

```
void openappletbox(void);
```

Open the applet manager window.

helptopic()

```
void helptopic(string & topic);
```

Calls up Windows help on the given topic.

helpwhatsthis()

```
void helpwhatisthis(void);
```

Puts OPW into “help what is this” mode - question mark help.

openregbox()

```
void openregbox(void);
```

Open the program registration window.

Choices Functions

openchoicesbox()

```
void openchoicesbox(void);
```

Opens the **Choices** dialogue box.

generalchoices()

```
void generalchoices(string &timeformat, string &dateformat, int
warnings, int help);
```

Controls the General choices on the **Choices** dialogue box *(see 15.5 in the Reference Guide)*.

The strings `timeformat` and `stringformat` specify **Date format** and **Time format**.

If `warnings` is non-zero, **Warnings** is selected.

If `help` is non-zero, **Load interactive help** is selected.

generalchoices2()

```
void generalchoices2(int globalclip, int nclipboards, int y, int z);
```

If bit 0 of `globalclip` is non-zero, then the global clipboard will be used. The second argument is the number of clipboards.

readgeneralchoices2()

```
int readgeneralchoices2(int n);
```

Returns the variable set by the n'th parameter of the `generalchoices2()` function. So `readgeneralchoices2(1)` returns the number of clip boards.

gettimeformat()

```
void gettimeformat(string &dateformat, string &timeformat);
```

Returns in `dateformat` and `timeformat` the current date and time format strings set in the **Choices** dialogue box.

undochoices()

```
void undochoices(int save, int buffsize);
```

Controls the **Undo** choices on the **Choices** dialogue box *(see 15.5 in the Reference Guide)*.

If `save` is non-zero, **Save buffer** is selected. The parameter buffsize is the **Buffer size** in bytes.

viewchoices1()

```
void viewchoices1(int update, int flash, int period, int rate, int
notused);
```

Controls the View choices on the **Choices** dialogue box *(see 15.6 in the Reference Guide)*.

If `update` is non-zero, **Instant updates on path edit** is selected.

If `flash` is non-zero, **Flashing caret** is selected. The parameter `period` is the **Period** in centi-seconds for a complete flash cycle, and `rate` is the fraction of the flash period that the caret is on for (where 0x10000 = unity).

viewchoices2()

```
void viewchoices2(int toolboxattached, int toolboxx, int toolboxy,
int infopalattached, int infopalx, int infopaly);
```

Controls the View choices on the **Choices** dialogue box *(see 15.6 in the Reference Guide)*.

If `toolboxattached` bit 0 is set, **Attach toolbox** is selected, and parameters `toolboxx` and `toolboxy` specify the absolute screen position of the toolbox. If bit 1 is set the toolbox is allowed off screen.

If `infopalattached` bit 0 is set, **Attach info palette** is selected, and parameters `infopalx` and `infopaly` specify the absolute screen position of the info palette. If bit 1 is set the info palette is allowed off screen.

viewchoices3()

```
void viewchoices3(int bits, int unused, int snapx, int snapy);
```

If bit 0 of `bits` is 1, then no drop shadow drawn around pages on pasteboard.

If bit 1 of bits is 1, then crosshairs are displayed when moving frames.

`snapx` and `snapy` specify in OS units the range for snapping. The default is 12 OS units. Snap distance is now independent of view magnification.

viewchoices4()

```
void viewchoices4(int x0, int x1, int y0, int y1);
```

Screen co-ordinates for opening windows.

viewchoices5()

```
void viewchoices5(int longclickdelay, int unused, int unused, int
unused);
```

Sets the time in centiseconds for separating short from long click drags.

textchoices()

```
void textchoices(string &fontname, int renamestyle, int quotes, int
inset, int overtype, int pcdelete, int renamecolour, int unused);
```

Controls the Text choices on the **Choices** dialogue box *(see 15.7 in the Reference Guide)*.

The parameter `fontname` is the **Single shift font** name.

If `renamestyle` is non-zero, **Rename styles** is selected.

If `quotes` is non-zero, **Smart quotes** is selected.

The parameter `inset` is the **Text frame inset** in 1/1000 pt.

If `overtype` is non-zero, **Overtype** is selected.

If `pcdelete` is non-zero, **PC-style delete** is selected.

If `renamecolour` is non-zero, **Rename colours** is selected.

`textchoicesbits()`

```
int textchoicesbits(void);
```

Returns bits specifying the settings of the Text choices in the **Choices** dialogue box:

Bit	Meaning when set
0	PC-style delete
1	Rename styles
2	Smart quotes
3	Overtype
4	Rename colours
5	Drag and drop
6	Save ignored words
7	Check for repeated words

`textchoices2()`

```
void textchoices2(int repeated words, int minlength, int spare2, int
spare3);
```

The first parameter controls if the spell checker will find repeated words. The second is the minimum length of words which will be spell checked.

`picturechoices()`

```
void picturechoices(int import, int aspect, int autoscale,
int centre, int inset, int runaround);
```

Controls the Picture choices on the **Choices** dialogue box *(see 15.8 in the Reference Guide)*.

The parameter `import` is the **Picture import with Ctrl** percentage.

If `aspect` is non-zero, **Aspect lock** is selected.

If `autoscale` is non-zero, **Auto scale** is selected.

If `centre` is non-zero, **Centre lock** is selected.

The parameter `inset` is the **Picture frame** inset in 1/1000 pt.

The parameter `runaround` is the **Runaround error** (where 0x10000 = unity).

`picturechoices2()`

```
void picturechoices2(int picturecachemax, int useproxylimit, int
proxylimit,int usereflimit, int reflimit);
```

`picturechoices3()`

```
void picturechoices3(int dither, int scale, int bpp, int bits);
```

Sets the defaults for creating proxy pictures. For scaling 0x10000 = 100%. The last argument is a bitfield variable, bit 0 of which controls if graphics files are checked for missing pictures when documents are

loaded.

linechoices()

```
void linechoices(int width, int capwidth, int capheight);
```

Controls the Line choices on the **Choices** dialogue box *(see 15.9 in the Reference Guide)*.

The parameter `width` is the **Default line width** in 1/1000 pt. A value of 0 means 'thin' i.e. the thinnest line width possible on the output device.

The parameter `capwidth` is the **Default triangle width** specified as a multiple of the line width (where 0x10000 = unity).

The parameter `capheight` is the **Default triangle height** specified as a multiple of the line width (where 0x10000 = unity).

printchoices()

```
void printchoices(int mode, int bleed, int thickness, int unused)
```

Controls miscellaneous printing defaults. If `mode` is zero, polling occurs during printing so that the **Printing** dialogue box is displayed. If `mode` is non-zero, no polling occurs and queueing is used. This latter setting may be used for background printing with a suitable printer driver.

The parameter `bleed` is the amount of page bleed used when printing with the **Bleed** option selected *(see 12.3 in the Reference Guide)*. The units are 1/1000 pt.

The parameter `thickness` is the thickness of the printers marks when printed with the **Printers marks** option selected *(see 12.3 in the Reference Guide)*. The units are 1/1000 pt.

printchoices2()

```
void printchoices2(int printbits_xor, int printbits_and, int spare,
int spare);
```

The bits of parameter `printbits` control miscellaneous printing defaults.

Bit	Meaning when set
0	Print pictures upside down. Use for printer drivers that invert images with OS_Sprite 56 when bit 0 of R3 is set.
1	Use grey scale sprite for output to mono printers.
2	Mail merge will produce a file for each mail version. If not set, will produce one print job containing all the merged versions. This bit is only applicable if the polling is used i.e. mode is 0 in printchoices().
3	Enable prescan pass on printer output.
4	Enable dialogue box that warns if output is bigger than paper size.
5	If this is set, the printer scale will be taken into account when calculating the 'flatness' of curved lines to use when rendering to the printer.
6	If this is set then when 'no pictures' is selected when printing a guide line will not be drawn around pictures.
7	Enable direct PostScript output.

To set bit 0, use xor = 1, and = ~1.
To clear bit 0, use xor = 0, and = ~1.

getprintbits()

```
int getprintbits(void);
```

Returns values of `printbits`, which is the first parameter of the function `printchoices2()`.

printchoices3()

```
void printchoices3(int crophlength, int crophstandoff, int
cropvlength, int cropvstandoff);
```

Sets the length and extreme ends of the horizontal and vertical crop marks in 1/1000 pt. Thickness of crop marks in set by `printchoices()`.

printchoices4()

```
void printchoices4(int reghlength, int regvlength, int regradius, int
reghstandoff, int regvstandoff);
```

Sets the length, radius and the distance to the centre of the horizontal and vertical registration marks in 1/1000pt. Thickness of registration marks in set by `printchoices()`.

postscriptchoices0()

```
void postscriptchoices0(int level,int fonttype,int generic,int spare);
```

`level` is the PostScript language level 1, 2 or 3

`fonttype` is the type of embedded fonts 1 or 3

`generic` is 1 for generic PostScript, or 0.

toolchoices()

```
void toolchoices(int xn, int yn, int spare1, int spare2);
```

If the toolbox is not attached, the parameters `xn` and `yn` specify the size of the toolbox.

toolrank()

```
void toolrank(int toolnumber, int rank, int bits, int spare1);
```

Specifies the position of the tool `toolnumber` on the toolbox. Parameter `rank` is the position number. If bit 0 of parameter `bits` is set, the tool is preselected. If bit 1 of parameter `bits` is set, the tool is not displayed on the toolbox.

generalpref()

```
void generalpref(int autosave, int interval, int prompt, int units,
int chapter, int separator);
```

Controls the General preferences on the **Preferences** dialogue box *(see 15.12 in the Reference Manual).*

If `autosave` is non-zero, **Auto save** is enabled with the interval specified in 1/100's of a second. If prompt is non-zero, **Prompt for auto save** is selected.

Parameter `units` specifies the **Document units**:

0 mm
1 inches
2 pts

Parameter `chapter` specifies the **Start chapter number**.

Parameter `separator` is split into two 16 bit values for the two characters making up the **Chapter/page separator**.

generalpref2()

```
void generalpref2(int autopageinsert,int autopagedelete,int
spare1,int spare2);
```

Controls autopage insertion and deletion.

generalpref3()

```
void generalpref3(int featurelevel,int spare0,int spare1,int spare2);
```

Default document feature level

viewpref1()

```
void viewpref1(int pasteboardx, int pasteboardy, int rulerx,
int rulery, int snapguides, int snapobjects, int guidepos);
```

Controls the View preferences on the **Preferences** dialogue box *(see 15.13 in the Reference Manual)*.

The parameter `pasteboardx` specifies the **Pasteboard width** in 1/1000 pt. The parameter `pasteboardy` specifies the depth of the pasteboard above and below the document.

Parameters `rulerx` and `rulery` specify **Ruler origin** X and **Ruler origin** Y in 1/1000 pt.

If `snapguides` is non-zero, **Snap to ruler guides** is selected.

If `snapobjects` is non-zero, **Snap to objects** is selected.

If `guidepos` is non-zero, **Guidelines to front** is selected, otherwise **Guidelines to back** is selected.

textpref()

```
void textpref(int spacing, int origin, int superscipt, int subscript,
int smallcaps, int tabchar);
```

Controls the Text preferences on the **Preferences** dialogue box *(see 15.14 in the Reference Manual)*.

Parameter `spacing` is the **Baseline grid spacing** specified in 1/1000 pt. The parameter origin is the **Baseline grid origin** specified in 1/1000 pt.

Parameters `superscript`, `subscript` and `smallcaps` specify **Superscript size**, **Subscript size** and **Small caps** size (where 0x10000 = 100%).

Parameter tabchar is the ASCII code for the **Decimal tab character**. It is redundant from version 2.51. See `textpref2()` below.

textpref2()

```
void textpref2(string &decimaltab, string &hyphenationchars, string
&spare1, string &spare2);
```

The `decimaltab` and `hyphenationchars` fields can take up to 4 characters which act as decimal tab characters and hyphenation characters respectively. The most significant characters (i.e. the ones used first) are on the left of the strings. Macro expansion is active on these strings allowing Unicode characters to be entered.

A null hyphenation character string makes the program default to using `'-'` ensuring compatibility with older versions of the program. The last argument of textpref(), previously set the single decimal tab character and is redundant from version 2.51 onwards.

File Menu Functions

openfileinfo()

```
void openfileinfo(void);
```

Opens the file info window for the current document. This effectivey replaces fileinfo_box().

opennewbox()

```
void opennewbox(void);
```

Opens the **New** dialogue box.

getnewpagesize()

```
int getnewpagesize(int widthorheight);
```

For use on the **New** and **Page guidelines** dialogue boxes.

setnewpagesize()

```
void setnewpagesize(int width, int height, string &name, int user);
```

For use on the **New** and **Page guidelines** dialogue boxes.

openfilesaveas()

```
void openfilesaveas(void);
```

Opens the save as window for the current document. This effectively replaces savedocument_box();

saveasstylesheet()

```
void saveasstylesheet(void);
```

Opens the save as stylesheet window for the current document.

openfileopen()

```
void openfileopen(void);
```

Opens the open file window.

savedocument()

```
void savedocument(string &name);
```

Saves the document with name specified. If `name` is a null string, uses the current name if it has been saved before, otherwise reports an error. See **`savedocumentastype()`** below.

openexportbox()
```
void openexportbox(void);
```
Opens the export window for the current selection in the current document.

openimportbox()
```
void openimportbox(void);
```
Opens the import window for the current document.

export_name()
```
int export_name(string &name);
```
Returns non-zero and the name of the type of object that may be saved, in string `name`. This function is used by ***Ovation Pro*** to display the correct menu text on the **File** menu.

reverttosaved()
```
void reverttosaved(void);
```
Restores the document to the most recently saved version. Equivalent to the **Revert to saved** option.

issaved()
```
int issaved(void);
```
Returns non-zero if the document has been saved. This function is used by ***Ovation Pro*** to shade **File➪Revert to saved** if it is unavailable.

ischanged()
```
int ischanged(void);
```
Returns non-zero if the document has been modified since it was last saved. This function is used by ***Ovation Pro*** to shade **File➪Revert to saved** if it is unavailable.

openprintbox()
```
void openprintbox(void);
```
Opens the **Print** dialogue box.

openprintmergebox()
```
void openprintmergebox(void);
```
Opens the **Mail merge** dialogue box.

saveasdefault()

```
void saveasdefault(void);
```

Saves the current document as the default document. Equivalent to the **Save as default** option.

savedocumentastype();

```
void savedocumentastype(string & name,int filetype);
```

Saves the current document. Over the existing function savedocument() this one offers the ability to save as DDL, save a stylesheet or to make use of any transsavers. filetype is the numeric filetype. For example ***Ovation Pro*** document 0xB27 ***Ovation Pro*** stylesheet 0xB26 DDL 0xB25. `name` can be the null string in which case the current document name is used.

Note that for DDL, stylesheet and transsaver saves there is no overwrite warning.

quit()

```
void quit(void);
```

Quits ***Ovation Pro***.

Edit Menu Functions

selectall()

```
void selectall(void);
```

Selects story if text frame active, otherwise selects all objects. Equivalent to the **Select all** option.

clearselection()

```
void clearselection(void);
```

Deselects selected text or objects. Equivalent to the **Clear** option.

cutselection()

```
void cutselection(void);
```

Deletes selected text, picture or object and transfers it to the clipboard. Equivalent to the **Cut** option.

cancopy()

```
int cancopy(void);
```

Returns non-zero if the selected object may be copied. For use on the **Edit** menu.

copyselection()

```
void copyselection(void);
```

Copies selected text, picture or object to the clipboard. Equivalent to the **Copy** option.

candelete()

```
int candelete(void);
```

Returns non-zero if the selected object may be deleted. For use on the **Edit** menu.

deleteselection()

```
void deleteselection(void);
```

Deletes selected text, picture or object. Equivalent to the **Delete** option.

copychapter()

```
void copychapter(void);
```

This function must only be used if the current view is set up.

pastechapter()

```
void pastechapter(int after);
```

The parameter `after` controls if the chapter is pasted after the current one. This function must only be used if the current view is set up.

selection_name()

```
string selection_name(void);
```

Returns the type of object currently selected. This function is used by Ovation Pro to display the correct object type after **Cut**, **Copy** and **Delete** on the **Edit** menu.

nselected()

```
int nselected(void);
```

Returns the number of objects selected. This function is used by Ovation Pro to shade the **Cut**, **Copy** and **Delete** options when there is nothing selected.

pasteselection()

```
void pasteselection(void);
```

Pastes contents of the clipboard into the document. Equivalent to the **Paste** option.

paste_name()

```
string paste_name(void);
```

Returns the type of object currently selected. This function is used by Ovation Pro to display the correct object type after **Cut**, **Copy** and **Delete** on the **Edit** menu.

embedobject()

```
void embedobject(void);
```

Embeds the object on the clipboard into the text at the caret. Equivalent to the **Embed object** option.

cliptype()

```
int cliptype(void);
```

Returns the type of object on the clipboard:

-1 Clipboard is empty
0 Clipboard contains an object
1 Clipboard contains picture
2 Clipboard contains text
3 Clipboard contains Windows clipboard contents
4 Clipboard contains a chapter

This function is used by ***Ovation Pro*** to shade the **Paste** and **Embed object** options when the clipboard is empty.

setcurrentclipboard()
```
int setcurrentclipboard(int index);
```
Set the current clipboard and return its index.

getcurrentclipboard()
```
int getcurrentclipboard(void);
```
Return the index of the current clipboard.

The following macro definitions are useful

```
defmacro(2,"nextclip","{setcurrentclipboard(getcurrentclipboard()+1)}","")
;
defmacro(2,"prevclip","{setcurrentclipboard(getcurrentclipboard()-1)}","")
;
defmacro(2,"bigcopy","{nextclip}{copyselection()}","");
defmacro(2,"bigpaste","{pasteselection()}{prevclip}","");
```

The first two provide a way of stepping forward and backward through the clipboards. `bigcopy` and `bigpaste` automatically step the clipboard after the operation, this allows you to do a series of copies in one document and then move to another document and do a series of pastes. These definitions can be used to replace the normal copy/paste operations or be bound to new keys.

undo()
```
void undo(void);
```
Undo the last operation. Equivalent to the **Undo** option.

canundo()
```
int canundo(void);
```
Returns non-zero if there is an operation that can be undone. This function is used by ***Ovation Pro*** to shade the **Undo** option when it is unavailable.

undostring()
```
string undostring(void);
```
Returns the name of the operation that can be undone. This string is displayed after the **Undo** option on the **Edit** menu.

redo()
```
void redo(void);
```
Redo the last operation. Equivalent to the **Redo** option.

canredo()

```
int canredo(void);
```

Returns non-zero if there is an operation that can be re-done. This function is used by ***Ovation Pro*** to shade the **Redo** option when it is unavailable.

redostring()

```
string redostring(void);
```

Returns the name of the operation that can be re-done. This string is displayed after the **Redo** option on the **Edit** menu.

openfindbox()

```
void openfindbox(void);
```

Opens the **Find** dialogue box. Equivalent to the **Find** option.

View menu functions

newview()
```
void newview(int master);
```
Opens a new view on the active document. Equivalent to the **New view** option.

openviewbox()
```
void openviewbox(void);
```
Opens the view options window for the current document.

gettools()
```
int gettools();
```
Returns non-zero if the toolbox is open, otherwise zero if it is closed.

settools()
```
void settools(int state);
```
If `state` is non-zero, open the toolbox, otherwise close the toolbox.

getbuttons()
```
int getbuttons();
```
Returns non-zero if the button bar is open, otherwise zero if it is closed.

setbuttons()
```
void setbuttons(int state);
```
If `state` is non-zero, open the button bar, otherwise close the button bar.

getinfopal()
```
int getinfopal(void);
```
Returns non-zero if the info palette is open, otherwise zero if it is closed.

setinfopal()
```
void setinfopal(int state);
```
If `state` is non-zero, open the info palette, otherwise close the info palette.

getrulers()
```
int getrulers(void);
```
Returns non-zero if the page rulers are displayed, otherwise zero if they are hidden.

setrulers()
```
void setrulers(int state);
```
If `state` is non-zero, display the page rulers, otherwise hide the page rulers.

getpasteboard()
```
int getpasteboard(void);
```
Returns non-zero if the pasteboard is displayed, otherwise zero if it is closed.

setpasteboard()
```
void setpasteboard(int state);
```
If `state` is non-zero, open the pasteboard, otherwise close the pasteboard.

getpictures()
```
int getpictures(void);
```
Returns non-zero if pictures are displayed, otherwise zero if they are hidden.

setpictures()
```
void setpictures(int state);
```
If `state` is non-zero, display pictures, otherwise hide pictures.

getguides()
```
int getguides(void);
```
Returns non-zero if guidelines are displayed, otherwise zero if they are hidden.

setguides()
```
void setguides(int state);
```
If `state` is non-zero, display guidelines, otherwise hide guidelines.

getmargins()
```
int getmargins(void);
```
Returns non-zero if print margins are displayed, otherwise zero if they are hidden.

setmargins()
```
void setmargins(int state);
```
If `state` is non-zero display print margins, otherwise hide print margins.

getfacingpages()
```
int getfacingpages(void);
```
Returns non-zero if the facing pages are displayed, otherwise zero if they are hidden.

setfacingpages()
```
void setfacingpages(int state);
```
If `state` is non-zero display facing pages, otherwise hide facing pages.

getinvisibles()
```
int getinvisibles(void);
```
Returns non-zero if the invisibles are displayed, otherwise zero if they are hidden.

setinvisibles()
```
void setinvisibles(int state);
```
If `state` is non-zero display invisibles, otherwise hide invisibles.

opengridbox()
```
void opengridbox(void);
```
Opens the grid dialogue box for the current document.

getgrid()
```
int getgrid(void);
```
Returns non-zero if the grid is displayed, otherwise zero if it is hidden.

setgrid()
`void setgrid(int state);`
If `state` is non-zero display the grid, otherwise hide the grid.

getbasegrid()
`int getbasegrid(void);`
Returns non-zero if the text baseline grid is displayed, otherwise zero if it is hidden.

setbasegrid()
`void setbasegrid(int state);`
If `state` is non-zero display the text baseline grid, otherwise hide the grid.

getgridlock()
`int getgridlock(void);`
`Bit 0 set` if lock to the object grid.
`Bit 1 set` if lock to the text base line grid.

setgridlock()
`void setgridlock(int state);`
Bit 0 of `state` is set lock to the object grid.
Bit 1 of `state` is set lock to the text base line grid.

openzoombox()
`void openzoombox(void);`
Opens the document zoom window for the current document.

setzoom()
`void setzoom(int opt, int mul, int div);`
Sets the document zoom. The parameter `opt` should be one of the following options which are equivalent to options on the **Zoom** dialogue box.

0	ZOOMFIT	Fit window
1	ZOOM50	50% zoom
2	ZOOM100	100% zoom
3	ZOOM200	200% zoom
4	ZOOMDEF	User defined variable zoom
5	ZOOMSCR	Fit screen
6	ZOOM75	75% zoom
7	ZOOM150	150% zoom
8	ZOOM400	400% zoom

If `opt` is set to ZOOMDEF, parameters `mul` and `div` should be set to the required zoom ratio, otherwise these parameters are not used.

getclipboard()

```
int getclipboard(void);
```

Returns non-zero if the clipboard is open, otherwise zero if it is closed.

setclipboard()

```
void setclipboard(int state);
```

If `state` is zero close the clipboard, otherwise open the clipboard. To toggle the clipboard open and closed, use:

```
setclipboard(!getclipboard());
```

Text/picture menu functions

openobjectcontentsbox()

```
void openobjectcontentsbox(void);
```

Opens the **Modify text** or **Modify picture** dialogue boxes. The actual box opened depends upon the currently active object.

opentextfont()

```
void opentextfont(void);
```

Opens the font menu.

getfont()

```
int getfont(void);
```

Returns the handle of the font at the caret or throughout the selection.

setfont()

```
void setfont(int handle);
```

Changes text to the font identified by `handle`.

getfontname()

```
int getfontname(int handle, string &name);
```

Returns in string `name`, the name of the font identified by `handle`. Returns non-zero if a font with that handle exists, otherwise returns zero. Font handles are numbered from 0 upwards.

fontschanged()

```
int fontschanged(void);
```

Returns non-zero if the list of available fonts has changed i.e. one or more fonts have been added or removed from the system. ***Ovation Pro*** uses this function to update the **Font** menu.

getfontsubname()

```
int getfontsubname(int fn, int sn, int bits, string &s)
```

Return in s the sub-name of the font specified by the family font name index `fn`, and the sub-font index sn. Returns 0 is `sn` is out of range.

Bit	Meaning when set
0	Add N, B, I, BI to font submenu to indicate which weights are mapped to Normal, Bold, Italic or Bold Italic.

getsubfont()

```
int getsubfont(void)
```

Returns the current sub-font index.

setsubfont()

```
void setsubfont(int font, int sub);
```

Set font and sub-font. The parameter sub should be set to -1 in order to set the normal variation of the font.

getfontsize()

```
int getfontsize(void);
```

Returns the size of the font at the caret or throughout the selection. Value returned is in 1/1000 pt.

setfontsize()

```
void setfontsize(int size);
```

Change text to font size specified by `size`. Value should be in 1/1000 pt.

geteffect()

```
int geteffect(void);
```

Returns an int whose bits specify which text effects are active at the caret or throughout the selection:

0x1	NORMAL
0x2	BOLD
0x4	ITALIC
0x8	UNDERLINE
0x10	REVERSE
0x20	SUPERSCRIPT
0x40	SUBSCRIPT

seteffect()

```
void seteffect(int effect);
```

Set text effect. The bits of parameter `effect` should be set according to the table given above.

getcase()

```
int getcase(void);
```

Returns the case of the text at the caret or throughout the selection:

0	Normal
1	Title
2	All caps

setcase()
```
void setcase(int case);
```
Set the case of text. The parameter `case` should be set according to the table above.

opencolourbox()
```
void opencolourbox(void);
```
Opens the text **Colour** dialogue box.

getulwidth()
```
void getulwidth(string &width);
```
Returns in the string `width`, the underline width.

For use on the underline **Width** menu.

setulwidth()
```
void setulwidth(string &width);
```
Sets the underline width to the value specified by `width`.

For use on the underline **Width** menu.

gettint()
```
int gettint(void);
```
Returns the current tint.

For use on the colour picker **Tint** menu.

settint()
```
void settint(int tint);
```
Sets the tint to the value specified by `tint`.

For use on the colour picker **Tint** menu.

openformatbox()
```
void openformatbox(void);
```
Opens the **Format** dialogue box.

getalign()
```
int getalign(void);
```
Returns the alignment of selected paragraphs:

0 Left aligned
1 Centred
2 Right aligned
3 Fully justified

setalign()
```
void setalign(int align);
```
Sets the alignment of selected paragraphs. The parameter `align` should be set according to the table above.

getleading()
```
void getleading(string &leading);
```
Returns in string `leading`, the leading of selected paragraphs. Typically the returned string will be of the form `20%`, `14pt` or `+6pt`.

setleading()
```
void setleading(string &leading);
```
Sets the leading of selected paragraphs. The parameter `leading` should be of the form `20%`, `14pt` or `+6pt`.

opentabsbox()
```
void opentabsbox(void);
```
Opens the **Tabs** dialogue box.

getparalevel()
```
int getparalevel(void);
```
Return current paragraph level 0..7

setparalevel()
```
void setparalevel(int level);
```
Set current paragraph level 0..7

setnumberformat()
```
void setnumberformat(int format);
```
Used to set the bullet/paragraph number format in the bullet window. The bottom 8 bits of format are split into 2 4 bit fields. The bottom 4 bits give the style of the number, Roman, Arabic etc. The top 4 bits set what number is displayed, current level, all levels etc.

getnumberformat()
```
int getnumberformat(void);
```
Used by the bullet window to get the bullet/paragraph number format.

getbaseshift()
```
int getbaseshift(void);
```
Returns the vertical shift of the text at the caret or throughout the selection (where 0x10000 = 100%).

setbaseshift()
```
void setbaseshift(int shift);
```
Sets the vertical shift of text (where 0x10000 = 100%).

getsmallcaps()
```
int getsmallcaps(void);
```
Returns the small caps state.

setsmallcaps()
```
void setsmallcaps(int state);
```
Set the small caps state.

caretcontext()
```
void caretcontext(void);
```
Used to set the internal cached values of style parameters to those at the caret.

openpicturereferencebox()
```
void openpicturereferencebox(void);
```
Opens the dialogue box for controlling picture references.

openproxybox()
```
void openproxybox(void);
```
Opens the dialogue box for controlling picture proxies.

Style menu functions

openstylebox()

```
void openstylebox(void);
```

Opens the **Edit style** dialogue box.

getscope()

```
int getscope(void);
```

For internal use by **Edit style** dialogue box only.

setscope()

```
void setscope(int scope);
```

For internal use by **Edit style** dialogue box only.

getuserstyle()

```
int getuserstyle(int handle, string &name, int mode);
```

Returns in string name, the name of the style identified by `handle`. Returns non-zero if a style with that handle exists, otherwise returns zero. Style handles are numbered from 0 upwards. For the **Style** menu, `mode` should be 1 this appends the associated key press. A value of 0 returns just the name, a value of 2 appends a tab followed by the key presss.

addstyle()

```
void addstyle(int handle);
```

Applies the style identified by `handle` to the text.

iscurrentstyle()

```
int iscurrentstyle(int handle);
```

Returns non-zero if the style identified by `handle` is current at the caret or throughout the selection.

getstylescope()

```
int getstylescope(int handle);
```

Returns the scope for the style specified by `handle`.

Value	Scope
0	Selection
1	Paragraph
2	Word
3	Line
4	Story

islocaleffect()

```
int islocaleffect(void)
```

Returns non-zero if there are local effects present at the caret or throughout the selection.

cleareffects()

```
void cleareffects(void);
```

Clears all local effects from the caret or selection.

getcurrentstyle()

```
int getcurrentstyle(int handle, string &name, int excluding);
```

Returns in string `name`, the name of the current style identified by `handle`. Returns non-zero if a style with that handle exists, otherwise returns zero. Current style handles are numbered from 0 upwards. The parameter `excluding` may have one of the following values:

0 Include BodyText
1 Exclude BodyText
2 Exclude BodyText if it is the bottom style

remstyle()

```
void remstyle(int handle);
```

Removes the style identified by `handle`, from the caret or selection.

isuserstyle()

```
int isuserstyle(int index);
```

Used by the style menu, returns true if the indexed style is present.

setuserstyle()

```
void setuserstyle(int index);
```

Used by the style menu, add the indexed style.

Font Functions

fontsetup()
```
void fontsetup(int fontbits, int spare, int spare, int spare);
```
All bits reserved.

openfontman()
```
void openfontman(void);
```
Opens the font manager window.

findglyph()
```
int findglyph(string & name);
```
Returns the code value for a given glyph name.

findglyphname()
```
string findglyphname(int code);
```
Returns the glyph name for a given code.

fontreplace()
```
void fontreplace(string & newidentifier,string & oldidentifier,int
spare);
```
This allows you to specify a font identifier and its replacement. For example;

fontreplace("Arial","Homerton.Medium",0);

fontremovereplace()
```
void fontremovereplace(string & newidentifier,string & oldidentifier);
```
This removes a replacement rule.

fontclearreplace()
```
void fontclearreplace(void);
```
Deletes all fontreplace() instructions.

fontreplacestate()
```
int fontreplacestate(int state);
```
Allows font replacement to be turned on or off, returns the previous state.

fontignore()
```
void fontignore(string & identifier,int spare);
```
This makes OP omit a Windows font from its font catalogue.

fontremoveignore()
```
void fontremoveignore(string & identifier);
```
Removes a fontignore() instruction.

fontclearignore()
```
void fontclearignore(void);
```
Deletes all fontignore() instructions.

fontignorestate()
```
int fontignorestate(int state);
```
Allows font ignoring to be turned on or off, returns the previous state.

fontmap()
```
void fontmap(string & newtypeface,string & newweight,string &
newidentifier,int newflags,string & existingfont,string * spare,int
spare1,int spare2);
```

`newtypeface` - font typeface name.
`newweight` - full font weight name (usually including the typeface name).
`newidentifier` - font name as saved in document file i.e. language independent name. This can be "" if it is the same as newweight.
`newflags` - bit 0 set for bold, bit 1 set for italic.
`existingfont` - Windows font name.
`spare` - not used at present.
`spare1` - not used at present.
`spare2` - not used at present.

For example;

```
fontmap("Homerton","Homerton Bold","Homerton.Bold",1,"Arial
Bold","",0,0);
```

fontremovemap()
```
void fontremovemap(string & member);
```
Removes the map instruction for this font.

fontclearmap()

```
void fontclearmap(void);
```

Deletes all fontmap() instructions.

fontmapstate()

```
int fontmapstate(int state);
```

Allows font mapping to be turned on or off, returns the previous state.

forcefontschange()

```
void forcefontschange(void);
```

Tells OPW to update its font catalogue.

Object menu functions

activetype()

```
int activetype(int objecttype);
```

Returns information about the type of object specified by `objecttype`. This should be set to

`TEXTFRAME` or `PICTFRAME`. It returns:

0	If no objects of this type are selected
1	If an object of this type is selected
>1	If the contents of the object are selected

canmodify()

```
int canmodify(void);
```

Returns non-zero if the selected object may be modified. For use on the **Object** menu.

openobjectbox()

```
void openobjectbox(void);
```

Opens the **Modify** dialogue box.

getguidewidth()

```
int getguidewidth(void);
```

Returns the column guide width.

For use on the column guide **Width** menu.

setguidewidth()

```
void setguidewidth(int width);
```

Sets the column guide width.

For use on the column guide **Width** menu.

getvalign()

```
int getvalign(void);
```

Returns the vertical alignment:

For use on the frame **Alignment** menu.

setvalign()

```
void setvalign(int align);
```

Sets the vertical alignment.

For use on the frame **Alignment** menu.

setfirstline();
```
void setfirstline(int firstline);
```
Sets the first baseline setting for text frames. For use on the **First line** menu.

Values of firstline

0 Accent
1 Ascender
2 Fit
3 Grid

getfirstline();
```
int  getfirstline(void);
```
Get the first baseline setting for text frames.

getlinewidth()
```
int getlinewidth(void);
```
Returns the line width.

For use on the line **Width** menu.

setlinewidth()
```
void setlinewidth(int width);
```
Sets the line width.

For use on the line **Width** menu.

getlinepattern()
```
int getlinepattern(void);
```
Returns the line pattern.

For use on the line **Pattern** menu.

getlinescale()
```
int getlinescale(void);
```
Returns the line scale. For use on the line **Pattern** menu.

setlinepattern()
```
void setlinepattern(int style, int scale);
```
Sets the line pattern.

For use on the line **Pattern** menu.

getlinejoin()
```
int getlinejoin(void);
```
Returns the line join.

For use on the line **Join** menu.

setlinejoin()
```
void setlinejoin(int join);
```
Sets the line join.

For use on the line **Join** menu.

getlinecaps()
```
int getlinecaps(int start);
```
Returns the line caps style.

For use on the line **Start caps** and **End caps** menus.

setlinecaps()
```
void setlinecaps(int start, int style);
```
Sets the line caps style.

For use on the line **Start caps** and **End caps** menus.

getlinetriangle()
```
int getlinetriangle(int start, int height);
```
Returns the line triangle caps size.

For use on the line caps **Width** and **Height** menus.

setlinetriangle()
```
void setlinetriangle(int start, int height, int size);
```
Sets the line triangle caps size.

For use on the line caps **Width** and **Height** menus.

getlinewindingrule()
```
int getlinewindingrule(void);
```
For internal use only.

setlinewindingrule()
```
void setlinewindingrule(int rule);
```
For internal use only.

openborderbox()
```
void openborderbox(void);
```
Opens the **Border** dialogue box.

opentextflowbox()
```
void opentextflowbox(void);
```
Opens the **Text flow** dialogue box.

openduplicatebox()
```
void openduplicatebox(void);
```
Opens the **Duplicate** dialogue box.

canduplicate()
```
int canduplicate(void);
```
Returns non-zero if the selected object may be duplicated. For use on the **Object** menu.

duplicateselection()
```
void duplicateselection(int n, int hshift, int vshift);
```
Duplicates the current object. If `n` < 0, does `n` duplications using dialogue box shifts. If `n` = 0, sets dialogue box offsets to `hshift` and `vshift`. If `n` > 0, does n duplications with offsets `hshift` and `vshift`.

canshape()
```
int canshape(void);
```
Returns non-zero if the selected object may be re-shaped. For use on the **Object** menu.

openshapebox()
```
void openshapebox(void);
```
Opens the **Shape** dialogue box.

cangofront()

```
int cangofront(void);
```

Returns non-zero if the selected object can be brought to the front. This function is used to shade the **Front** option when it is unavailable.

objectfront()

```
void objectfront(void);
```

Brings the selected object to the front. Equivalent to the **Front** option.

cangoback()

```
int cangoback(void);
```

Returns non-zero if the selected object can be sent to the back. This function is used to shade the **Back** option when it is unavailable.

objectback()

```
void objectback(void);
```

Sends the selected object to the back. Equivalent to the **Back** option.

cangroup()

```
int cangroup(void);
```

Returns non-zero if there are a number of selected objects that may be grouped. This function is used to control the **Group** option.

groupselection()

```
void groupselection(void);
```

Groups the selected objects. Equivalent to the **Group** option.

canungroup()

```
int canungroup(void);
```

Returns non-zero if the selected objects may be ungrouped. This function is used to control the **Ungroup** option.

ungroupselection()

```
void ungroupselection(void);
```

Ungroups the selected objects.

canmakelocal()

```
int canmakelocal(void);
```

Returns non-zero if the selected object can be made local. This function is used to control the **Make local** option.

makeselectionlocal()

```
void makeselectionlocal(void);
```

Makes selected master object local. Equivalent to the **Make local** option.

canmakemaster()

```
int canmakemaster(void);
```

Returns non-zero if the selected object can be made back into a master object. This function is used to control the **Make master** option.

makeselectionmaster()

```
void makeselectionmaster(void);
```

Makes localised object back into a master object. Equivalent to the **Make master** option.

snaptogrid()

```
void snaptogrid(void);
```

Snaps the selected object to the grid. Equivalent to the **Snap to grid** option.

objectforward()

```
void objectforward(int n);
```

Brings the selected object forward by n layers.

objectbackward()

```
void objectbackward(int n);
```

Sends the selected object backward by n layers.

getsnapstate()

```
int getsnapstate(void);
```

Returns snap state for the current document.

Bit	Meaning when set
0	snap to frames
1	snap to ruler guidelines

setsnapstate()

```
void setsnapstate(int bits)
```

Set snap state for the current document.

Bit	Meaning when set
0	snap to frames
1	snap to ruler guidelines

getselectionx()

```
int getselectionx(void);
```

getselectiony()

```
int getselectiony(void);
```

getselectionw()

```
int getselectionw(void);
```

getselectionh()

```
int getselectionh(void);
```

setselectionarea();

```
void setselectionarea(int x,int y,int w,int h);
```

Functions for obtaining the selection position (`x` and `y`) and dimensions (`w` and `h`) and setting them. These parallel the info palette display for the select tool, so the values are for the first selected object, not for the overall selection.

Page menu functions

A document has `n` chapters, indexed 0..(`n` - 1), and a chapter has `m` pages, indexed 0..(`m` - 1).

opennewchapterbox()

```
void opennewchapterbox(int pos);
```

Creates a new chapter and opens the **Modify chapter** dialogue box. If `pos` is non-zero the new chapter is inserted after the current chapter, otherwise it is inserted before.

openmodchapterbox()

```
void openmodchapterbox(void);
```

Opens the **Modify chapter** dialogue box.

ismasterview()

```
int ismasterview(void);
```

Returns non-zero if the master pages are open. This function is used to control various options on the **Page** menu.

openpageguidebox()

```
void openpageguidebox();
```

Opens the **Page guidelines** dialogue box.

deletechapter()

```
void deletechapter(void);
```

Deletes the current chapter.

openinspagebox()

```
void openinspagebox(void);
```

Opens the **Insert page** dialogue box.

opendelpagebox()

```
void opendelpagebox(void);
```

Opens the **Delete page** dialogue box.

opengotopagebox()
```
void opengotopagebox(void);
```
Opens the **Goto page** dialogue box.

documentchapters()
```
int documentchapters(int file);
```
Returns the total number of chapters in the file specified.

chapterpages()
```
int chapterpages(int chapter, int file);
```
Returns the total number of pages in the chapter and file specified.

chapterstart()
```
int chapterstart(int file);
```
Returns the chapter start number for the file specified.

pagestart()
```
int pagestart(int chapter, int file);
```
Returns the page start number for the chapter and file specified.

selectionpage()
```
int selectionpage(void);
```
Returns the page index of the page containing the selection or -1 if there is no selection.

selectionchapter()
```
int selectionchapter(void);
```
Returns the chapter index of the chapter containing the selection or -1 if there is no selection.

currentpage()
```
int currentpage(int view);
```
Returns the index of current page i.e. the page in the ‘middle’ of the view specified.

currentchapter()
```
int currentchapter(int view);
```
Returns the index of current chapter i.e. the chapter in the ‘middle’ of the view specified.

gotopage()

```
void gotopage(int page, int chapter, int view);
```

Go to the page and chapter in the view specified.

chapterbits()

```
int chapterbits(int chapter, int file);
```

Returns chapter information for the chapter and file specified.

Bit	Meaning when set
0	Chapter starts on the left
1	Chapter is double sided
2	Page start number is specified
3	Count blank pages at end of chapters when numbering
4..7	Page number format 2 decimal 1 Roman lower 0 Roman upper

maxpagewidth()

```
int maxpagewidth(int file);
```

Return width of widest page in document (in 1000/pt).

maxpageheight()

```
int maxpageheight(int file);
```

Return height of tallest page in document (in 1000/pt).

Misc menu functions

checkstory()

```
void checkstory(int caret);
```

Spell check current story. If `caret` is non-zero the spell check commences from the caret position, otherwise from the start of the story. Equivalent to the **Check from caret** and **Check story** options.

checkword()

```
void checkword(void);
```

Spellcheck current word. Equivalent to the **Check word** option.

checkdocument()

```
void checkdocument(void);
```

Spellcheck entire document. Equivalent to the **Check document** option.

getcheck()

```
int getcheck(void);
```

Returns non-zero if the **Check as you type** option is selected.

setcheck()

```
void setcheck(int asyoutype);
```

Selects the **Check as you type** option.

opendictionarybox()

```
void opendictionarybox(void);
```

Opens the **Dictionary** dialogue box.

getuserdict()

```
int getuserdict(int handle, string &name);
```

Returns in string `name`, the name of the dictionary whose handle is specified by `handle`. Returns non-zero if a dictionary with that handle exists, otherwise returns zero.

setuserdict()
```
void setuserdict(int handle);
```
Attaches the dictionary specified by `handle`.

isuserdict()
```
int isuserdict(int handle);
```
For use by the **Dictionary** dialogue box only.

getspellerrors()
```
int getspellerrors(void);
```
Returns non-zero if spellcheck errors are currently being displayed, otherwise returns 0.

setspellerrors()
```
void setspellerrors(int state);
```
If `state` is non-zero, display spelling errors, otherwise hide spelling errors.

opencharsbox()
```
void opencharsbox(void);
```
Opens the **Characters** box.

openmacrobox()
```
void openmacrobox(void);
```
Opens the **Macros** dialogue box.

flushundobuffer()
```
void flushundobuffer(void);
```
Flushes the undo buffer. Equivalent to the **Misc➪Clear undo buffer** option.

datestring()
```
string datestring(void);
```
Returns the current date.

timestring()
```
string timestring(void);
```
Returns the current time.

openmergebox()

```
void openmergebox(void);
```

Opens the **Insert➪Merge** tag dialogue box.

openprefbox()

```
void openprefbox(void);
```

Opens the **Preferences** dialogue box.

getunits()

```
int getunits(void);
```

Returns an integer value representing the current units:

```
0    mm
1    inches
2    pts
```

setunits()

```
void setunits(int unit);
```

For use by the **Preferences** dialogue box only.

openimportwordsbox()

```
void openimportwords(void);
```

Opens the import words window -used by dictionary edit window and menu.

openexportwordsbox()

```
void openexportwords(void);
```

Opens the export words window -used by dictionary edit window and menu.

Picture functions

pixsetup()

```
void pixsetup(int fontbits, int jpegbits, int spritebits, int spare2);
```

Controls various aspects of picture plotting. This function is used to set defaults in the file `<OPD>\AutoRun\!Custom.csc`

All bits reserved.

olesetup()

```
void olesetup(int olebits, int spare1, int spare2, int spare3);
```

All bits reserved.

openpictureinfobox()

```
void openpictureinfobox(void)
```

Opens the picture information dialogue box.

Colour Functions

getselectioncolour()

```
int getselectioncolour(int type, int info)
```

Return colour information about the selected object of the type specified. The argument `info` determines the type of information returned.

Info	Meaning
0	Return colour key
1	Return tint (0x10000 = 100%)
2	Return overprint status

Return zero if there is no object of that type selected.

setselectioncolour()

```
void setselectioncolour(int type, int key, int tint, int overprint);
```

Set the colour, tint or overprint status of the selected object of the type specified, to the colour specified by `key`.

If `key` <= 0, the colour will not be changed. If `tint` < 0, the tint will not be changed.
If `overprint` < 0, overprint status will not be changed.

getcolourpal()

```
int getcolourpal(int i, int tint, int spare, int file);
```

Returns an BBGGRR00 palette word for the colour indexed by `i` with the tint specified (where 0x10000 = 100% and -1 = transparent).

openeditcolourbox()

```
void openeditcolourbox(int key);
```

Opens the **Edit colour** dialogue box with the colour specified by `key` selected. A value of 0 gives the behaviour of versions before 2.45, the first colour selected.

getcolour()

```
int getcolour(int i, string &name, int file);
```

Returns `key` for `i`'th colour or 0 if colour does not exist. Also returns in string `name`, the name of the colour. If `file` is 0 then the function works as in versions before 2.45. Note that the colour keys are unique and repeatable, e.g. Cyan always has the same key.

setcolour()

```
void setcolour(int handle);
```

For use by the **Edit colour** dialogue box only.

getcolourchart()

```
int getcolourchart(int handle, string &name);
```

Returns in string `name`, the name of the colour chart whose handle is specified by `handle`. Returns non-zero if a colour chart with that handle exists, otherwise returns zero.

setcolourchart()

```
void setcolourchart(int handle);
```

For use by the **Colour chart** dialogue box only.

iscurrentchart()

```
int iscurrentchart(int handle);
```

For use by the **Colour chart** dialogue box only.

Document Tagged Data Manager

This allows applets to embed information inside documents.

tagwrite()

```
int tagwrite(int file, int tag, int value, int index);
```

Write the integer value into the `file` specified. The integer is identified by `tag[index]`, which must be unique integers. Returns `tag`.

tagwrites()

```
int tagwrites(int file, int tag, string &value);
```

Write the string `value` into the `file` specified. The string is identified by `tag`, which must be a unique integer. Returns `tag`.

tagexists()

```
int tagexists(int file, int tag, int index);
```

Returns 1 if `tag[index]` exists in the file specified.

tagread()

```
int tagread(int file, int tag, int index);
```

Returns `tag[index]` value from the file specified.

tagreads()

```
int tagreads(int file, int tag, string &s);
```

Reads tag string value and returns 1 if tag exists.

Tag values should be allocated by taking the standard applet 3 letter prefix as a binary number and shifting it left 8 places - (AAA<<8). This gives each applet 256 tags.

Printing Functions

printdocument()

```
void printdocument(void);
```

Prints the current document without opening the **Print** dialogue box.

prprints()

```
int prprints(string &s);
```

Outputs string `s` to the printer file.

prinfo()

```
int prinfo(string &name);
```

Sets `name` to the name of the current printer. Returns the R0 value of PDriver_Info or -1 if there is no printer driver.

printerwidth()

```
int printerwidth(void);
```

Return width of printable area in 1000/pt, otherwise returns a value less than 0 if there is no printer.

printerheight()

```
int printerheight(void);
```

Return height of printable area in 1000/pt, otherwise returns a value less than 0 if there is no printer.

paperwidth()

```
int paperwidth(void);
```

Return width of paper in 1000/pt, otherwise returns a value less than 0 if there is no printer.

paperheight()

```
int paperheight(void);
```

Return height of paper in 1000/pt, otherwise returns a value less than 0 if there is no printer.

papermarkx()

```
int papermarkx(int marks);
```

Returns the X dimension of the printers mark specified by the `marks` parameter (see imposition functions).

papermarky()
```
int papermarky(int marks);
```
Returns the Y dimension of the printers mark specified by the `marks` parameter (see imposition functions).

getclipcurve()
```
int getclipcurve(int n);
```
This returns element `n` of the clip curve.

prpagew()
```
int prpagew(void);
```
Returns the width of the page being printed (only valid after `EVENT_PRSTARTPAPER`).

prpageh()
```
int prpageh(void);
```
Returns the height of the page being printed (only valid after `EVENT_PRSTARTPAPER`).

prworkw()
```
int prworkw(void);
```
Returns the overall width of the page being printed including bleed etc. (only valid after `EVENT_PRSTARTPAPER`).

prworkh()
```
int prworkh(void);
```
Returns the overall height of the page being printed including bleed etc. (only valid after `EVENT_PRSTARTPAPER`).

prworkmaxh()
```
int prpageh(void);
```
Returns the maximum page height in the print job including bleed etc. (valid at any time).

prworkmaxw()
```
int prworkmaxw(void);
```
Returns the maximum page width in the print job including bleed etc. (valid at any time).

prpagex()
```
int prpagex(void);
```
Returns the X coordinate of the top left corner of the page currently being rendered.

prpagey()
```
int prpagey(void);
```
Return the Y coordinate of the top left corner of the page currently being rendered.

prpageshiftx()
```
int prpageshiftx(void);
```
Returns the X coordinate of the bottom left hand corner of the page rendering coordinate system relative to the bottom left hand corner of the paper.

prpageshifty()
```
int prpageshifty(void);
```
Returns the Y coordinate of the bottom left hand corner of the page rendering coordinate system relative to the bottom left hand corner of the paper.

prareax0()
```
int prareax0(void);
```

prareay0()
```
int prareay0(void);
```

prareax1()
```
int prareax1(void);
```

prareay1()
```
int prareay1(void);
```
The above functions return coordinates of a rectangle in hypothetical page space.

openfields()
```
void openfields(void);
```
Opens the **Merge fields** dialogue box

field()
```
string field(int n);
```
Returns the n'th field from the original csv file.

nextrecord()
```
int nextrecord(void);
```
Steps to the next csv record. Returns non-zero at eof.

maxfield()

```
int maxfield(void);
```

Returns maximum number of csv fields in this record.

ps_level()

```
int ps_level(void);
```

Return the current PostScript level.

getprintername()

```
int getprintername(int n,string & name);
```

Puts the n’th printer name in the string variable and returns non-zero, or returns 0.

setprintername()

```
void setprintername(string  & name);
```

Sets the named printer as the current one.

Imposition Functions

Crop and registration marks are defined by a `marks` value, the bits of which are as follows;

Bit	Meaning when set
0	Top registration circle
1	Bottom registration circle
2	Left registration circle
3	Right registration circle
4	Left/top vertical crop mark
5	Left/bottom vertical crop mark
6	Left/top horizontal crop mark
7	Left/both horizontal crop mark
8	Right/top vertical crop mark
9	Right/both vertical crop mark
10	Right/top horizontal crop mark
11	Right/both horizontal crop mark

Information from the print dialogue box is passed by a `paperbits` value, the bits of which are as follows;

Bit	Meaning when set
0..3	Print 0 = odd pages 1 = even pages 2 = all pages
4	Sideways
5	Bleed

The print format menu shows formats in the order in which they are created (usually the order the scripts defining them execute). Clicking on the menu references the formats by their creation index `(printmodeindex)`. However internally the program uses a different unique number `(keyn)` for print formats. All menus and associated functions reference formats by `printmodeindex`.

getprintmode()

```
int getprintmode(void)
```

Returns current printmode index.

setprintmode()

```
void setprintmode(int printmodeindex)
```

Sets printmode index.

paperformat()

```
int paperformat(int printmodeindex, string &name);
```

This function returns zero if the printmode index does not exist.

paperaddformat()

```
void paperaddformat(string &formatname, int keyn, string &prname,
string &prnamecount, string &prnamesetup);
```

Add new imposition format called `formatname` with a unique key, `keyn`. The keys already defined are as follows:

Keyn	Format
0	Normal
1	Pamphlet
2	Galley
3	Printers Pairs
4	Booklet
5	Thumbnails
6	Tile
7	8-up Sheetwork
8	8-up Work & Turn

For every format, three functions must be supplied, following the prototypes below. Conventionally, the names given below are amended to use a unique identifier (e.g. `prthumb()`, `prthumbcount()` and `prthumbsetup()` for the thumbnail format).

prname()

```
int prname(int file, int keyn, int reason, int arg1, int arg2, int
arg3);
```

Returns general purpose requests about the layout.

`reason` = 0
`arg1` = max width
`arg2` = max height
`arg3` = not used

Return the maximum pages per sheet of paper.

`reason` = 1
`arg1` = max width
`arg2` = max height
`arg3` = not used

Call `papermaxarea()` with the max area of the pages.

Call `paperarea()` with the max crop mark area.

This is used to generate the scale values in the print dialogue box when scale to fit is enabled. Note these values are only for guidance. Accurate scaling to fit is carried out when the pages are printed.

`reason` = 2
`arg1` = max width
`arg2` = max height
`arg3` = not used

A new print mode has been set from the dialogue box. Return bits:

Bit	Meaning when set
0	Format is double sided
1	Format sets scale
2	Format sets orientation
3	No scale to fit
4	No print margins

`reason` = 3
`arg1` = max width
`arg2` = max height
`arg3` = not used

Return new print mode scale.

`reason` = 4
`arg1` = max width
`arg2` = max height
`arg3` = not used

Return new print mode orientation.

`reason` = 5
`arg1` = page bits i.e. bit 1= left hand page
`arg2` = page width
`arg3` = page height

Call `paperarea()` with the area of the marks.

Call `papermaxarea()` with the area of the pages.

Call `paperpagearea()` with the visible area of the page.

This is used to generate the print margin on screen. Return 0.

`reason` = 6
`arg1` = not used
`arg2` = not used
`arg3` = not used

Save local variables to document. Return 0.

`reason` = 7
`arg1=` not used
`arg2=` not used
`arg3=` not used

Are local variables different to those in file?

Return 0 if they are the same, otherwise 1 if different

`reason` = 8
`arg1` = max width
`arg2` = max height
`arg3` = not used

Load the print format values from the document. Called when the dialogue box is opened or a document is about to be printed.

Return bits (as reason code 2).

`reason` = 9
`arg1` = max width
`arg2` = max height
`arg3` = not used
The print dialogue box is about to be opened. Chance to calculate local variable values.

Return 0.

`prnamecount()`

```
int  prnamecount(int file, int keyn, int paperbits);
```

Return number of pieces of paper. Typically using `paperfirstpage()` and `papernextpage()` to scan through the pages to be printed.

`prnamesetup()`

```
void prnamesetup(int file, int keyn, int paperbits, int paperno, int
nopaper);
```

Generate the positions of the pages on this piece of paper. Typically this will call `paperpage()`, `paperarea()`, `paperpagearea()` to set out the pages on the paper.

Imposition Support functions

paperfirstpage()

```
void paperfirstpage(void);
```

Set up for a scan through pages.

papernextpage()

```
int papernextpage(void);
```

Used to scan through pages. Returns bits set as follows:

Bit	Meaning when set
31	Print this page i.e. you don't print all pages
30	Page returned i.e. you stop if this is not set
...	
0	Left hand page

paperpage()

```
void paperpage(int n, int xpo, int ypo, int xlo, int xhi, int yhi,
int ylo, int angle);
```

Used to position a page on the paper. where `n` is an index running from 0..max number of pages per piece of paper. `angle` is in degrees, only multiples of 90 make sense.

paperpagescale()

```
void paperpagescale(int n, int xpo, int ypo, int xlo, int xhi, int
yhi, int ylo, int angle,int xscale,int yscale);
```

As paperpage() but the page can be scaled. For the scale parameters 0x10000 == 100%.

paperpagearea()

```
void paperpagearea(int n, int xlo, int xhi, int yhi, int ylo, int
marks);
```

Set up crop marks for current page.

paperarea()

```
void paperarea(int xlo, int xhi, int yhi, int ylo, int marks);
```

Used to specify the border of what has been placed on the paper. This is the area that crop marks go around i.e. the area of the pages, excluding the bleed.

The importance of this area is that it specifies where the colour separation info goes. Just possibly all the crop marks can be generated for each page.

papermaxarea()

```
void papermaxarea(int xlo, int xhi, int yhi, int ylo);
```

Used to specify the max area of what *might* be on the paper. The problem this solves, is for example printing pamphlet mode with fit or centre enabled. Some pieces of paper will have only one page on them, we don't want that single page centring - rather an area equal to 2 pages must be centred.

paperpagewidth()

```
int paperpagewidth(void);
```

Returns width of current page.

paperpageheight()

```
int paperpageheight(void);
```

Returns height of current page.

paperbleed()

```
int paperbleed(void);
```

Returns current bleed, i.e. 0 (no bleed) or the bleed size.

paperxscale()

```
int paperxscale(void);
```

Returns X scale i.e. from print dialogue box.

paperyscale()

```
int paperyscale(void);
```

Returns Y scale e.g. from print dialogue box.

paperbits()

```
int paperbits(void);
```

Returns current paper bits.

papermarks()

```
int papermarks(int fp);
```

Returns current printers marks i.e. 0 or the marks bits.

papercount()

```
int papercount(int parity);
```

Returns the paper count for the print job. the argument parity can be 0, 1 or -1. -1 just counts the number of pages. 0 or 1 match their left or right-handedness and insert any necessary blank pages.

Window Functions

create_window()

```
int  create_window(string &templatename);
```

Creates the window specified by `templatename` and returns the window handle.

delete_window()

```
void delete_window(int handle);
```

Deletes the window specified by `handle`.

display_window()

```
void display_window(int handle, int offset, int bits);
```

Displays the window specified by `handle`. If offset is non-zero, multiple copies of the window will be offset so that they do not overlay one another. The parameter `bits` controls where the window is opened:

Bit	Function when set
0	Open window centred on screen
1	Open window at front

When you call `display_window()`, an OPEN WINDOW event (0x03) will be generated. This gives you the chance to alter the position of the window before passing on the values to `open_window()`.

open_window()

void open_window(int handle, int x0, int y0, int x1, int y1, int scx, int scy, int bhandle);

Opens the window specified by `handle`.

Values for bhandle;
-1 open the window at the top
-2 open the window at the bottom
-3 no change in window stack position
-4 after the topmost non-topmost window as a topmost window
-5 after the topmost non-topmost window

close_window()

void close_window(int handle);

Closes the window specified by `handle`.

extent_window()

void extent_window(int handle, int x0, int y0, int x1, int y1);

Sets the window extent to `x0`, `y0` and `x1`, `y1`.

refresh_area()

```
void refresh_area(int update, int handle, int x0, int y0, int x1, int
y1);
```

Refresh window area specified.

refresh_icon()

```
void refresh_icon(int update, int handle, int icon);
```

These let you generate redraws. The first parameter is the difference between a forced redraw and an update. i.e. setting off from scratch or from what is already there.

`refresh_icon(update, handle,-1)` is a special case and refreshes the whole visible window area.

addwindowhandler()

```
void addwindowhandler(int event, int handle, string &fnname);
```

Add event handler for the window specified by handle. When the event occurs, the handler fnname is called. Events supported are:

0 MOUSE CLICK

```
void clickfn(int handle, int icon, int bbits, int mx, int my);
```

1 KEY PRESS

```
int  keyfn(int handle, int icon, int key);
```

Two distinct types of key code may appear. A `key` value in the range 0 to 0xFFFF is a typed character. Other classes of this event occur when any key is held down. The `key` value then consists of a Windows virtual key code in the lower byte and various flags which may be set;

```
#define KEYVSHIFT (1<<24)
#define KEYVCTRL  (1<<25)
#define KEYVIRT   (1<<26)
#define KEYVALT   (1<<27)
#define KEYVWIN   (1<<28)
```

For a character key like ‘A’ you will see the key down event and then a typed character event. For a key like Delete you will see only the key down event. The `KEYVIRT` flag is always set for key down events - this is how the event can be distinguished from a typed character event.

2 CLOSE WINDOW

```
void closefn(int handle);
```

3 OPEN WINDOW

```
void openfn(int handle, int x0, int y0, int x1, int y1, int scx, int
scy, int bhandle);
```

This is called as a result of the display_window() function.

4 REDRAW

```
void redrawfn(int handle, int x0, int x1, int y0, int y1, int scx,
int scy, int gx0, int gx1, int gy0, int gy1);
```

If you don’t define events, reasonable default actions take place e.g. close the window.

5 OPEN WINDOW POST

```
void openfn(int handle, int x0, int y0, int x1, int y1, int scx, int
scy, int bhandle);
```

The window has been opened by Windows with the parameter values when this function is called.

6 DELETE WINDOW

```
void deletefn(int handle);
```

The window has been deleted. This may occur as a result of the parent being deleted, or the delete_window() function.

7 PARENT CLOSE

```
void parentclosefn(int handle);
```

The window's parent has closed.

writeicon()

```
void writeicon(int handle, int icon, string &s);
```

Write string s into the icon specified by icon, in the window specified by handle.

readicon()

```
string readicon(int handle, int icon);
```

Return string from the icon specified by icon, in the window specified by handle.

delete_icon()

```
void delete_icon(int handle, int icon);
```

Delete icon from window specified by handle.

select_icon()

```
void select_icon(int handle, int icon, int select);
```

Make icon selected.

shade_icon()

```
void shade_icon(int handle,int icon,int state);
```

Shade or unshade an icon, `state` true means shade.

set_icon_state()

```
void set_icon_state(int handle, int icon, int value, int mask);
```

Set the state of the icon in the window specified.

Newflags = (oldflags bic mask) eor value

insert_caret()

```
void insert_caret(int handle, int icon);
```

Insert caret into icon.

find_caret()

```
int find_caret(int item);
```

Returns the position of the focus. The parameter `item` takes the following values;
0 window
1 icon
2 caret/marking start position, this is the number of character positions
3 marking end position, this is the number of character positions
The last two values only make sense for edit controls. The function returns -1 if the value does not exist e.g. if no window has the focus. Examples;

```
w=find_caret(0); // return window with focus
i=find_caret(2); // position of caret in edit control
```

setactive()

```
void setactive(int handle);
```

Activates the given window.

setfocus()

```
void setfocus(int handle);
```

Gives the focus to the given window.

setfocusicon()

```
void setfocusicon(int handle,int icon)
```

Gives the focus to the given icon.

write_slice_icon()

```
void write_slice_icon(int handle,int icon,int param,string & value);
```

Allows you to manipulate abx controls. param is an abx tag like ‘s’ for a string, string is the new value. See the AppletSDK.

read_slice_icon()

```
int read_slice_icon(int handle,int icon,int param,string & value);
```

Reads a tag value from an abx control. Returns non-zero if the tag is present.

create_icon()

```
int  create_icon(int handle, int x0, int y0, int x1, int y1, int
flags, string &name, int maxlen, string &valid);
```

Beware this can (for indirected icons) allocate memory, and this will only be freed if the delete_icon() or delete_window() functions are used.

plot_icon()

```
void plot_icon(int x0, int y0, int x1, int y1, int flags, string
&name, int maxlen, string &valid);
```

Often if you want to redraw lots of varying icons, it is better to not actually create/delete icons, but to just plot them.

create_windowparent()

```
int create_windowparent(string & name,int parent);
```

Creates a child window. Windows works best when windows are children of one another.

geticon()

```
int geticon(int window,int icon,int coord);
```

Returns coordinates of an icon. The value of coord controls which coordinate is returned. 0 is x0, 1 is y0, 2 is x1, 3 is y1.

setwindowflags()

```
void setwindowflags(int handle,int bic,int orr);
#define WINDOWPANE 0x8
#define WINDOWFIXED 0x10
#define WINDOWNOTDIALOGUE 0x20
#define WINDOWNOSCROLLCHILD 0x40
#define WINDOWBEINGOPENED 0x80
#define WINDOWMINMAX 0x100
#define WINDOWAUTOCENTRE 0x200
```

Allows window flags to be twiddled. The bic argument is the bit to clear, and the orr argument the new value for it. The only value of user interest is WINDOWPANE which is used to make windows behave like tool windows.

Bookmark Functions

Bookmarks are invisible markers that may be positioned and moved around within text stories. Typically they are used by scripts which need to scan through stories, locating and replacing text strings. Note that bookmarks are retained in documents when they are saved. If this is not desired, then bookmarks must be explicitly deleted when the script terminates.

The extension applet BookMark demonstrates the use of many of these functions.

bmcreate()

```
int bmcreate(string &name);
```

Creates a bookmark with the name specified, and returns a handle for that bookmark.

bmdelete()

```
void bmdelete(int bm);
```

Deletes bookmark bm.

bmprevchar()

```
int bmprevchar(int bm);
```

Returns character before bookmark bm otherwise -1 if the bookmark is at the start of the story.

bmchar()

```
int bmchar(int bm);
```

Returns character at bookmark bm otherwise 0 if the bookmark is at the end of the story.

setbmtocaret()

```
void setbmtocaret(int bm);
```

Sets bookmark bm to the current caret position.

setcarettobm()

```
void setcarettobm(int bm);
```

Sets caret position to bookmark bm.

setbmtozone()

```
void setbmtozone(int bm1, int bm2);
```

Sets bookmarks bm1 and bm2 to the start and end of the selected zone.

setzonetobm()

```
void setzonetobm(int bm);
```

Selects zone between bookmark bm and the caret.

bmmove()

```
int bmmove(int bm, int direction, int unit);
```

Moves bookmark `bm` forwards or backwards by the amount specified. Returns 0 if end of text is reached.

If direction = 0 then the amount moved is determined by the parameter unit:

0 prev char
1 start of word
2 start of line
3 start of para
4 start of text
5 start of previous word
6 start of previous line
7 start of previous para
8 previous line

If direction = 1 then the amount moved is determined by the parameter unit:

0 next char
1 end of word
2 end of line
3 end of para
4 end of text
5 start of next word
6 start of next line
7 start of next para
8 next line

bmview()

```
void bmview(int bm);
```

Brings bookmark `bm` into view.

setbmtobm()

```
void setbmtobm(int bm1, int bm2);
```

Sets position of bookmark `bm1` to that of bookmark `bm2`.

bmcmp()

```
int bmcmp(int bm1, int bm2);
```

Compares position of bookmarks `bm1` and `bm2`. Return -1, 0 or 1 depending on whether bookmark `bm1` is before, at the same place or after bookmark `bm2`.

bmsearch()

```
int bmsearch(int bm1, int bm2, string &s, int flags);
```

Searches for string `s`, optionally starting from `bm1`. Return 0 if string is found, with `bm1` set to start of found string and `bm2` set to end of it. The bits in parameter flags refer to the standard search options in the **Find** dialogue box:

Bit	Function when set
0	Case sensitive search
1	Whole word search
2	All stories (not supported)
3	Find from start (not supported)
4	Wildcards supported

bmstartscan()

```
void bmstartscan(void);
```

Prepare to scan all stories in document.

bmnextstory()

```
int bmnextstory(int bm);
```

Moves `bm` to start of next story. Returns non-zero when no more stories.

bmname()

```
int bmname(int n, string &name);
```

Returns handle and name of bookmark number `n`, or returns 0 for no more bookmarks.

bmfind()

```
int bmfind(string &name);
```

Returns handle for bookmark `name`.

bmpage()

```
int bmpage(int bm);
```

Returns the index of the page containing the bookmark bm.

bmchapter()

```
int bmchapter(int bm);
```

Returns the index of the chapter containing the bookmark bm.

bmsection()

```
int bmsection(int bm);
```

Exactly the same function as `bmchapter()`.

The next few functions were originally supplied in the enhanced bookmark manager applet written by R.J.E. Thompson. They are now contained in the main program. The idea is that applets can declare a bookmark name prefix as hidden, any bookmarks with a hidden prefix should be ignored by other applets. For example the bookmark applet does not show them on the menus it presents to users. Applets use hidden bookmarks for their own internal purposes. Typically the prefix is the three letter identifier used by the applet for its variables etc. followed by an underscore character.

bm_hidden()

```
int bm_hidden(string &prefix)
```

Returns true if the prefix is hidden.

bm_hide()

```
void bm_hide(string &prefix)
```

Defines the prefix as hidden

bm_unhide()

```
void bm_unhide(string &prefix)
```

Removes the prefix from the list of hidden prefixes

bm_toggle()

```
void bm_toggle(string &prefix)
```

Toggles the hidden state of the prefix

bm_hiddenmark()

```
int bm_hiddenmark(string &name)
```

Given a bookmark name, returns 1 if it is hidden.

Miscellaneous functions

caretchar()

```
int caretchar(void);
```

Return the ASCII code for the character after the caret, or -1 if there is no active text frame or if the caret is at the end of a story.

type()

```
void type(string &s);
```

Types the string `s` at the current caret position. In addition to the usual printable characters, the string may contain control codes preceded by `|`, escape sequences preceded by `\`, macros enclosed in braces `{}` and system variables enclosed in angle brackets `<>` *(see 17.5 in the Reference Guide).*

```
type(s);
type("Hello World");
type("Beep|G");
type("Hello World\r");
type("{ActiveDate}");
type("<%USERNAME%>");
```

curpageno()

```
int curpageno(void);
```

Returns the current page number.

curchapterno()

```
int curchapterno(void);
```

Returns the current chapter number.

setmaindictionary()

```
void setmaindictionary(string mainname);
```

Sets the name of the default main dictionary. This function is used in `AutoRun.!Custom`.

setuserdictionary()

```
void setuserdictionary(string username);
```

Sets the name of the default user dictionary. This function is used in `AutoRun.!Custom`.

stringdp()

```
int stringdp(string &value);
```

Returns the string `value` representing a floating point number converted to an integer. The inverse of this function is `dpstring()`. These functions allow the script language, which only has integer variables, to handle floating point numbers. The conversion makes 1.00 = 0x10000.

```
// In this example i = 3686400 = 56.25 * 0x10000
int i = stringdp("56.25");
```

dpstring()

```
string dpstring(int value, int decimal);
```

Returns `value` converted to a string representing a floating point number. Parameter decimal specifies the number of decimal places required. This function is the inverse of `stringdp()`.

```
// In this example s = 56.3 (rather than 56.2 due to rounding).
string s = dpstring(3686400 , 1)
```

stringu()

```
int stringu(string &value, int unit);
```

Returns the string `value` converted to 1/1000 pt. The string may be fractional and optionally followed by the units; `mm`, `in`, `pt` etc. If the string `value` does not specify the units, the parameter `unit` will be used to determine them:

```
0    mm
1    in
2    pt
3    picas
4    centimetres
5    didot
6    cicero
7    foot
8    yard
```

The inverse of this function is `ustring()`. These functions allow the script language to handle floating point values in different units.

```
// In this example i = 36000
int i = stringu("0.5", 1);
```

ustring()

```
string ustring(int value, int unit);
```

Returns `value` in 1/1000 pt, converted to a string. The units specified by `unit` are appended to the string. This function is the inverse of `stringu()`.

```
// In this example s = 0.5in
string s = ustring(36000, 1);
```

picktool()

```
void picktool(int tool, int bits);
```

Selects the tool whose number is specified by `tool`. If bit 0 of parameter `bits` is set, the tool will persist.

fileinfo()

```
int fileinfo(int file, string &s);
```

Sets `s` to the filename for the file specified, and returns an integer whose bits give information about the file.

Bit	Meaning when set
0	File has been saved
1	File has been modified
2	File is readable
3	File is writable
4	File is special i.e. internal use not a document

filefromview()

```
int filefromview(int view);
```

Returns a file handle from a view handle.

currentview()

```
int currentview(void);
```

Returns the current view handle.

currentfile()

```
int currentfile(void)
```

Returns the current file handle

selectionfile()

```
int selectionfile(void)
```

Return the file handle of the file containing the selection , otherwise 0 if there is no selection.

setcurrentview()

```
int setcurrentview(int view);
```

Attach script to the view specified and return the previous view. The best form of usage is:

```
temp = setcurrentview(view);
...
< code using functions which depend upon the current view >
...
setcurrentview(temp);
```

setcurrentfile()

```
int setcurrentfile(int file);
```

Attach script to the file specified. Operation is similar to `setcurrentview()` described above. Note that setting the current view is more precise than setting the file because a file may have more than one view.

nextfile()

```
int nextfile(int file);
```

Returns next file handle - if called with 0 returns first file.

nextuserfile()

```
int nextuserfile(int file);
```

This is identical to `nextfile()` but skips any files which have special uses. Using `nextfile()` many applets have to check for special files like the clipboard by name, and given multiple clipboards, the picture cache document and so on, this is clumsy.

nextview()

```
int nextview(int view, int file);
```

Return next view handle for `view` and `file` specified, otherwise returns 0 if there is no next view. If called with a `view` of 0, returns first view for file.

viewhandle()

```
int viewhandle(int view);
```

Returns the window handle for a given view. Typically used when you want the applet window to be the child of the main window.

activeview()

```
int activeview(void);
```

Returns the view handle for the active view in ***Ovation Pro***.

defaultfile()

```
int defaultfile(void);
```

Returns the handle of the default document.

filescrap()

```
void filescrap(int file);
```

Remove file from memory.

loadfile()

```
int loadfile(string &name);
```

Load specified file, returning the file handle.

viewrepaint()

```
void viewrepaint(int view, int file);
```

Repaint the specified `file` and `view`.

If `file` is specified, repaint all views for the file.

If `view` is specified, just repaint that view.

If both `file` and `view` are null, repaint all views for all files.

password()

```
int password(string &readpw, string &writepw, int flags, int file);
```

Sets read password, write password or read-only flag for specified file. Note that the write password overrides the read one, so it's only necessary to enter a write password. The bits of parameter `flags` have the following meaning:

Bit	Meaning when set
0..2	
	= 0 Do nothing, just return value
	= 1 Unlock file
	= 2 Change passwords
3	readpw is valid
4	writepw is valid
5	read only bit is valid
8	file read only bit

Returns

Bit	Meaning when set
0	Read password set
1	Write password set
2	File is readable
3	File is writable
4	Read unlocked
5	Write unlocked
8	File is read-only

Bit 4 is always redundant because it always follows the readable bit.

Passwords can only be written on a file which is writable.

dcsfile()

```
int dcsfile(int file);
```

Opens the **Discard/Cancel/Save** dialogue box and returns 0 if the user clicks on **Discard**, otherwise returns non-zero. The file specified is discarded after a successful save.

prevchar()

```
int prevchar(void);
```

Returns the ASCII code for the character before the caret. Otherwise returns 0 if the caret is at the start of the story, or -1 if there is no caret.

replace()

```
int replace(string &s, int flags);
```

Replaces the selected text with the string `s`, and returns 0 if successful. The bits of parameter `flags` have the following meaning:

Bit	Function when set
0	Case sensitive search. The string is replaced in exactly the same format as specified.
1	Whole word search
2	All stories
4	Find from start
5	Wildcards supported

gettimestring()

```
void gettimestring(string &format, string &result);
```

Returns in `result` the date or time formatted according to the parameter `format`. The format string should be in the usual Acorn form. This function is useful because it allows you to get just one part of the date or time e.g.

```
gettimestring("%CE%YR", s);
n = stoi(s); // n==1996
```

zoneorder()

```
int zoneorder(void);
```

Returns < 0 if the selected zone has been selected right to left, > 0 for left right and 0 for no selected zone.

translate()

```
void translate(string &s);
```

Takes the string `s` of translatable text, and translates it. The string will be translated by having any tokens replaced by the corresponding text from the Messages file.

getfiletimestring()

```
void getfiletimestring(string &format, string &result, int file);
```

Returns in `result` the last time the specified file was saved to disc, formatted according to the parameter `format`. `file` is a document file handle.

gettimestamp()

```
void gettimestamp(string &format, string &result, string &path);
```

Returns in `result` the time stamp of the file specified by `path`, formatted according to the parameter `format`.

impexbits()

```
int impexbits(int filetype, int bic, int orr);
```

Sets the import/export bits for the filetype specified. Returns the new value of the import/export bits for the given filetype, otherwise returns -1 if the filetype is not known.

Bit	Meaning when set
0	Load this filetype
1	Set type to be Picture
2	Set type to be Text

In particular, bit 0 controls if files are loaded.

programversion()

```
int programversion();
```

Returns version of ***Ovation Pro*** multiplied by 100 e.g. 245 for version 2.45.

scale()

```
int scale(int value, int mul, int div);
```

Performs an accurate `(val * mul) / div` calculation where the intermediate result can be 64 bits.

checkstring()

```
int checkstring(int file, int language, string &word);
```

Returns non-zero if word exists in spell check dictionaries.

impexcopy()

```
int impexcopy(int basetype, int newtype, string &name);
```

This allows a new filetype to be added which is based on an existing file type. The typical use is adding support for variations on Draw files. Returns 0 on success.

ddl()

```
void ddl(string &code);
```

A function which executes a string of DDL code at the current selection. This is potentially very powerful, allowing script programs to create objects and generate text with effects.

autoscroll()

```
void autoscroll(int range, int a0, int a1, int a2);
```

This function controls the auto scrolling effect. All the parameters are fractions 0x10000=1.0.

The first parameter is the fraction of the window width or height, which the pointer must be inside for auto scrolling to take place. By default this is 0x2000 or 1/8th.

If the distance into the scroll zone is x pixels which is a fraction z of the zone size, then the amount scrolled is

```
shift=x*(a0+z*a1+z*z*a2)
```

By default:

a0=0x1000
a1=0x4000
a2=0x8000

transload()

```
int transload(string & from,int from_type,string & to,int to_type);
```

Translate file, returns 0 on success.

appletpath()

```
void appletpath(string & path);
```

Fills in the location of an applet. Typically this is used in an applet's initialisation code, if it needs access to files inside its folder.

setcharselection()

```
void setcharacterselection(int index);
```

Set character set in Characters window.

getcharselection()

```
int getcharacterselection(int index,string & name);
```

Get character set name in Characters window from index. Returns 0 when there are no more.

iscurrentcharselection()

```
int setcharacterselection(int index);
```

Return true if index is current character set in Characters window.

General Purpose Input Window

gpbox1()

```
void gpbox1(string & title,string & caption,string value,string
function);
```

function prototype

```
void function(int code,string value);
```

This is a general purpose user input window, it displays a window with a title bar, a writable icon and caption icon. When the user clicks on OK or Cancel the function will be called, the parameter code will have value 1 if OK is clicked, or 2 if Cancel is used, it may have the value 0 if the window is closed in some other way.

gpbox2()

```
void gpbox2(string & title,string & caption1,string & caption2,string
& value1,string & value2,string & function);
```

function prototype

```
void function(int code,string value1,string value 2);
```

Akin to gpbox1() but with two values.

gpbox4()

```
void gpbox4(string & title,string & caption1,string & caption2,string
& caption3,string & caption4,string & value1,string & value2,string &
value3,string & value4,string & function);
```

function prototype

```
void function(int code,string value1,string value2,string
value3,string value4);
```

Akin to gpbox1() but with four values.

Old and Obsolete Functions

The following are functions in the RISC OS version of ***Ovation Pro*** which are not appropriate to Windows. Some of them have been removed, some have partial functionality, others do nothing. Better ways of doing what these functions do are described in the rest of this manual.

`osclis()`
Removed (see `oscommand()`)

`swi13()`
Removed

`swin4()`
Removed

`swin8()`
Removed

`fx()`
fx 138 is the only case supported. See `keypress()`.

`bbc_adval()`
Does nothing

`bbc_inkey()`
Only the parameter values -1 (shift pressed) and -2 (control pressed) are supported. See `isctrl()`.

`bbc_vdu()`
Only the value 7 - beep is supported. See `beep()`.

`keyboardmode()`
Does nothing

`create_menuwindow()`
`delete_menuwindow()`
`fileinfo_box()`
`savedocument_box()`
`export_box()`
`savestylesheet_box()`
`view_box()`
`grid_box()`
`zoom_box()`
This group of functions are concerned with windows which are attached to the menu tree on RISC OS. They do nothing on the Windows version of ***Ovation Pro***. Alternative functions to open the relevent dialogue boxes are provided.

Colour Separations Applet Functions

csreadfrequency()

```
int csreadfrequency(void);
```

Returns the density of the half-tone screen for the current separation.

csreadangle()

```
int csreadangle(void);
```

Returns the half-tone screen angle for the current separation.

csreadpsscreen()

```
int csreadpsscreen(void);
```

Returns non-zero if a PostScript screen is being specified for the current separation.

csreadspot()

```
int csreadspot(void);
```

Returns a value representing a half-tone dot shape for the current separation.

csreadseps()

```
int csreadseps(void);
```

Returns non-zero if separations being printed.

csreadcolour()

```
int csreadcolour(string &name);
```

Returns in `s` the name of the separation currently being printed.

csreadpxfrequency()

```
int csreadpxfrequency(void);
```

Returns the density of the half-tone screen for the current picture frame (or the current values if the picture has no screen set up).

csreadpxangle()

```
int csreadpxangle(void);
```

Returns the half-tone screen angle for the current picture frame (or the current values if the picture has no screen set up).

csreadpxpsscreen()
```
int csreadpxpsscreen(void);
```
Returns non-zero if a PostScript screen is being specified for the current picture frame.

csreadpxspot()
```
int csreadpxspot(void);
```
Returns a value representing a half-tone dot shape for the current picture frame (or the current values if the picture has no screen set up).

csgetink()
```
int csgetink(void);
```
Returns the index of the current ink file.

csgetcurve();
```
int csgetcurve(void);
```
Returns the index of the current correction table file.

cschoices()
```
void cschoices(string &curvename, string &inkname. int rgbcorrection,
int cmykcorrection);
```
Controls the Colour choices on the **Choices** dialogue box.

If `rgbcorrection` is zero, RGB colours are not corrected when separated to CMYK. If `rgbcorrection` is 1, RGB to CMYK colour correction is achieved using the correction table specified by `curvename`.

If `cmykcorrection` is zero, CMYK colours are not corrected when displayed on-screen as RGB. If `cmykcorrection` is non-zero, CMYK to RGB correction is achieved using the ink data table specified by `inkname`.

cschoices2()
```
void cschoices2(int cstextx1, int cstextx2, int cstexty, int
cstextsize, string &cstextfont);
```
Specifies the position, font and size of the separation name (usually printed at the top of each separation).

cschoices3()
```
void cschoices3(int csbarx, int csbary, int csbarwidth, int
csbarheight);
```
Specifies the position and size of the colour bar (usually printed at the bottom of each separation).

cschoices4()

```
int cschoices4(int bit_orr, int bit_bic, int spare, int spare);
```

Returns new bits value.

```
newvalue=(oldvalue BIC bit_bic) OR bit_orr
```

#define CSCHOICEKNOCKOUT 0x1

When this bit is set, graphics with process colours will knockout on spot plates.

Default is knockout bit set, older versions of the program default to the opposite behaviour.

Event Handling Functions

You can add event handlers to call your own script functions when certain events take place.

addeventhandler()

```
void addeventhandler(int eventno, int user, string &function);
```

remeventhandler()

```
void remeventhandler(int eventno, int user, string &function);
```

These functions add and remove event handlers for events specified by eventno. When the event occurs, the event handler function is called with the user parameter `user`.

Each event supported is listed below, together with the exact format of the event handler function.

0x01 EVENT_FOPEN

```
void fileopen(int user, int file);
```

Issued when the document specified by file, has been opened.

0x02 EVENT_FCLOSE

```
void fileclose(int user, int file);
```

Issued when the document specified by file, has been closed.

0x03 EVENT_RUNPROGRAM

```
void runprogram(int user);
```

Issued when ***Ovation Pro*** has been started.

0x04 EVENT_ENDPROGRAM

```
void endprogram(int user);
```

Issued when ***Ovation Pro*** has been quit.

0x05 EVENT_VOPEN

```
void viewopen(int user, int view);
```

Issued when the view specified by view, has been opened.

0x06 EVENT_VCLOSE

```
void viewclose(int user, int view);
```

Issued when the view specified by view, has been closed.

0x07 EVENT_FPRESAVE

```
void presave(int user, int file, string &s);
```

Issued when document specified by handle file, is about to be saved. Parameter **s** is the name of the document, and can be changed if required. Setting it to the null string will stop saving taking place.

0x08 EVENT_FONTSCHANGED

```
void fontschanged(int user);
```

Issued when the available fonts have changed.

0x0e EVENT_OBJECTSELECTED0
0x0f EVENT_OBJECTSELECTED1
0x10 EVENT_OBJECTDESELECTED0
0x11 EVENT_OBJECTDESELECTED1

```
void sfn(int user, int file, int object);
```

Where there are 0 and 1 variants, the 0 comes before and the 1 after - if there is a non-subscripted variant that usually happens in the middle (of the event).

0x19 EVENT_RUNPROGRAM2

```
void runprogram2(int user);
```

Issued immediately after EVENT_RUNPROGRAM.

0x1a EVENT_MODIFYSELECTION
0x1b EVENT_TRANSFORMSELECTION
0x1c EVENT_TEXTFLOWSELECTION
0x1d EVENT_SELECTIONDELETED
0x1e EVENT_MASTERSELECTION
0x1f EVENT_LOCALSELECTION

```
void sfn(int user, int file);
```

Where there are 0 and 1 variants, the 0 comes before and the 1 after - if there is a non-subscripted variant that usually happens in the middle (of the event).

0x24 EVENT_ZOOM

```
void zoomchanged(int user, int file, int view);
```

Issued when the zoom of the specified view and file has changed.

0x2c EVENT_PREFERENCES

```
void preferenceschanged(int user, int view);
```

Issued when the Preferences for the document specified by handle view, have changed.

0x50 EVENT_NEWDOC0
0x5f EVENT_NEWDOCF
0x60 EVENT_BAREDOC0
0x6f EVENT_BAREDOCF
0x70 EVENT_LOADSTYLE0
0x7f EVENT_LOADSTYLEF

```
void createevent(int user, int file);
```

Issued when new documents are created, loading a DDL file or loading a style sheet.

In each case, there are 16 events from 0 to F, but only the first and last are currently supported i.e. before anything has happened to after everything has happened. Note that with event 0, there may be no filename defined.

0x91 EVENT_CHOICES

```
void choiceschanged(int user);
```

Issued when the application Choices have changed.

0x92 EVENT_MODIFYSELECTION0
0x93 EVENT_MODIFYSELECTION1
0x94 EVENT_TRANSFORMSELECTION0
0x95 EVENT_TRANSFORMSELECTION1
0x96 EVENT_TEXTFLOWSELECTION0
0x97 EVENT_TEXTFLOWSELECTION1
0x99 EVENT_TRANSFORMSELECTIONR0
0x9a EVENT_TRANSFORMSELECTIONR1
0x9b EVENT_SHAPESELECTION0
0x9c EVENT_SHAPESELECTION1
0x9d EVENT_MOVESELECTION0
0x9e EVENT_MOVESELECTION1

```
void sfn(int user, int file);
```

Where there are 0 and 1 variants, the 0 comes before and the 1 after - if there is a non-subscripted variant that usually happens in the middle (of the event).

0x100 EVENT_PRGOPRINTER0
```
void printfn(int user, int view);
```
Issued immediately before VDU output is redirected to the printer.

0x101 EVENT_PRGOPRINTER1
```
void printfn(int user, int view);
```
Issued immediately after VDU output is redirected to the printer.

0x102 EVENT_PRGONORMAL0
```
void printfn(int user, int view);
```
Issued immediately before VDU output is redirected to the screen.

0x103 EVENT_PRGONORMAL1
```
void printfn(int user, int view);
```
Issued immediately after VDU output is redirected to the screen.

0x104 EVENT_PRSTARTPAGE
```
void printfn(int user, int view);
```

0x105 EVENT_PRENDPAGE
```
void printfn(int user, int view);
```

0x106 EVENT_PRSTARTCOPY
```
void printfn(int user, int view);
```

0x107 EVENT_PRENDCOPY
```
void printfn(int user, int view);
```

0x108 EVENT_PRSTARTRECORD
```
void printfn(int user, int view);
```
Issued when a mail merge record for the view specified is just about to be printed.

0x109 EVENT_PRENDRECORD
```
void printfn(int user, int view);
```
Issued when a mail merge record for the view specified has just finished printing.

0x10a EVENT_PRSTARTJOB
```
void printfn(int user, int view);
```
Issued when the print output file is opened.

0x10b EVENT_PRENDJOB
```
void printfn(int user, int view);
```
Issued when the print output file is closed.

0x10c EVENT_PRSTART
```
void printfn(int user, int view);
```
Issued when the print job for the view specified is about to start.

0x10d EVENT_PREND
```
void printfn(int user, int view);
```
Issued when the print job for the view specified has ended.

0x10e EVENT_PRCACHE0
```
void printfn(int user, int view);
```
Issued before fonts are cached.

0x10f EVENT_PRCACHE1
```
void printfn(int user, int view);
```
Issued after fonts have been cached.

0x110 EVENT_PRCOPIES
```
void printfn(int user, int view);
```
Issued immediately after asking the WIMP how many copies are to be printed.

0x111 EVENT_PRSTARTSEPARATION
```
void printfn(int user, int view);
```

0x112 EVENT_PRENDSEPARATION
```
void printfn(int user, int view);
```

0x113 EVENT_PRSTARTPICT
```
void printfn(int user, int view);
```
Issued before a picture is printed.

0x114 EVENT_PRENDPICT
```
void printfn(int user, int view);
```
Issued after a picture has been printed.

0x115 EVENT_PRCHANGE
```
void printfn(int user, int view);
```
Issued when printer driver starts up.

0x116 EVENT_PRPAGECHANGE
```
void printfn(int user, int view);
```
Issued when printer configuration has changed.

0x118 EVENT_PRSTARTPAPER
```
void printfn(int user, int view);
```
Issued when printing starts on a new piece of paper.

0x119 EVENT_PRENDPAPER
```
void printfn(int user, int view);
```
Issued when printing ends on a piece of paper.

0x11a EVENT_PRCOPIES2
```
void printfn(int user, int view);
```
Issued immediately after EVENT_PRCOPIES.

0x11B EVENT_PRPXSTART
0x11C EVENT_PRPXEND
```
void printfn(int user, int view);
```
These events occur just after the clip rectangle for a picture is set and just before it is cleared. They are only generated for printer output.

0x136 EVENT_SETCLIPBOARD
```
void clipfn(int user,int oldcurrentclipboard,int newcurrentclipboard);
```
Raised when the current clipboard changes.

0x137 EVENT_SETNOCLIPBOARD

```
void clipfn(int user,int oldnumber,int newnumber);
```

Raised when the number of clipboards changes.

Note that changing the number of clip boards can also generate the SETCLIPBOARD event, because if the current clip board is number 6 and the total is set to 3, the current clipboard will change to number 2. In such a case, SETCLIPBOARD will be issued before SETNOCLIPBOARD.

0x152 EVENT_CHOICESSAVE

```
void savechoicesfn(int user);
```

Issued when the application Choices have been saved.

0x153 EVENT_PREFERENCESSAVE

```
void saveprefsfn(int user, int view);
```

Issued when the Preferences for the document identified by view have been saved.

0x154 EVENT_COLOURCHANGED

```
void cfn(int user, int file);
```

Issued when the colour definitions are changed.

0x210 EVENT_FILEMODIFIED

```
void modfile(int user, int file);
```

Issued when the document specified by file, has been modified.

0x211 EVENT_FILESAVED

```
void filesaved(int user, int file);
```

Issued when the document specified by file, has been successfully saved.

0x212 EVENT_CARET

```
void caretevent(int user, int view, int gain);
```

Issued when the view specified by view, gains or loses the input focus. Parameter gain is 1 if the input focus is gained, or 0 if it is lost.

0x213 EVENT_VIEWCLICK

```
int viewclick(int user, int view, int mousex, int mousey, int
mousebuttons);
```

Issued when a mouse click occurs. Parameters `mousex` and `mousey` are the mouse coordinates and `mousebuttons` identifies the type of click according to the table below. Return the value of `mousebuttons`, or `0` to prevent anything from happening.

VBCLICKS	0x4
VBCLICKM	0x2
VBCLICKA	0x1
VBDRAGS	0x40
VBDRAGM	0x20
VBDRAGA	0x10
VBDBLCLICKS	0x400
VBDBLCLICKM	0x200
VBDBLCLICKA	0x100
VBLONGCLICKS	0x40000
VBLONGCLICKM	0x20000
VBLONGCLICKA	0x10000
VBTRIPCLICKS	0x400000
VBTRIPCLICKM	0x200000
VBTRIPCLICKA	0x100000
VBQUADCLICKS	0x4000000
VBQUADCLICKM	0x2000000
VBQUADCLICKA	0x1000000
VBPENTCLICKS	0x40000000
VBPENTCLICKM	0x20000000
VBPENTCLICKA	0x10000000

0x214 EVENT_VOPENPOST

```
void viewopen(int user, int view);
```

Issued after the view specified by view, has been opened.

0x215 EVENT_ACTIVEVIEW

```
void activeview(int user, int view, int gain);
```

Issued when the tools/info palette are attached to a new window. Generated once with 0 and once with 1 for 'gain'.

0x216 EVENT_CURRPAGE

```
void activeview(int user, int view);
```

Issued when the current page changes.

0x300 EVENT_KEYPRESS

```
int keyfn(int user, int view, int key);
```

Issued when a key is pressed in the view specified. Return a new key value or -1 to claim the key press.

0x301 EVENT_INSTEXT0
0x302 EVENT_INSTEXT1

```
int textfn(int user, string &s, int claimed);
```

EVENT_INSTEXT0 is issued before a string is entered into a document. `EVENT_INSTEXT1` is issued after a string is entered into a document.

The string `s` can be changed if required (only sensible for `EVENT_INSTEXT0`).

This function should return a non-zero value to stop the string being added (again only sensible for `EVENT_INSTEXT0`).
Note that this function is passed a string of characters being entered into the document because of 'type ahead'. In general individual characters are not added to the document.

Bear in mind that there may be pending styles present in the text, so it may be judicious to use the `type()` function only for `EVENT_INSTEXT1` events.

The `claimed` parameter is the result of other functions linked to this event. If it is non-zero, ignore the string, and return claimed.

0x303 EVENT_FONTMISSING

```
int fontmissingfn(int user, int file, string &fontname, int claimed);
```

Generated when missing fonts are found during document load. Return 1 if you have done something about it else 0.

This event is issued when a document is loaded and fonts are missing. If, after issuing this event, fonts are still missing, the standard warning box is displayed.

0x304 EVENT_TEXTUPDATE

```
void textupdate(int user, int file);
```

Generated when the text at the caret changes style etc. This may include the caret vanishing e.g. if you change tool.

Note that you can't use `pause()` in this event handler.

0x305 EVENT_USERSTYLECHANGE

```
void stylechangefn(int user, int file);
```

Generated when the user styles have changed i.e. after **OK** in the **Edit style** dialogue box.

0x306 EVENT_BEEP

```
int beepfn(int user, int claimed, int code);
```

Generated when ***Ovation Pro*** beeps in various situations specified by code:

0x100 Spelling error
0x101 Spelling OK
0x102 Text find fails (search and replace)

Return 1 to claim or 0 to pass on.

0x307 EVENT_FILENOREAD

```
int filenoreadfn(int user, int file, int claimed);
```

Document is going to be removed from memory because there is no read permission. Return 1 to claim event and keep the document.

0x308 EVENT_FILEWRITEFAIL

```
int filewritefailfn(int user, int file, int claimed);
```

Attempt to modify a document failed because no write permission.

0x309 EVENT_FILESCRAP

```
int filescrapfn(int user, int file, int claimed);
```

Document is going to be removed from memory because all views have been closed. Return 1 to claim event and keep the document.

0x30a EVENT_NONRECTCLIP

```
int clipfn(int user, int claim);
```

Asks if non-rectangular clipping is available. Return `claim` = 1 if it is.

0x30b EVENT_DOCLIP

```
void doclipfn(int user);
```

Asks for clip area to be set up.

0x30C EVENT_SPELLCHECK

```
int spellfn(int user, int file, int language, int spare, string
&word);
```

If a word can't be found in the dictionaries, this event is generated. Return non-zero to claim - i.e. to say that the word is in fact correct. Typical use is dealing with 'foreign' languages where words are split by single quotes e.g. aujourd'hui.

0x310 EVENT_MERGESTART
0x311 EVENT_MERGEFINISH
0x312 EVENT_MERGEDATA
0x313 EVENT_MERGESTRIP
0x314 EVENT_MERGECLEAR

All these are related to mail merging. The start event is issued when a mail merge is about to start on a file. The function prototype is;

```
void startfn(int user, int file, string &name);
```

Where `name` is the name of the CSV file, if there is one, otherwise it is a null string.

The DATA event is issued when new data is merged in. The STRIP event is when the mail merge tags are being removed i.e. someone clicked on 'fix'. Finally the CLEAR event is issued when mail merge data is being removed from the document e.g. at the end of a mail merge session.

The data prototype is;

```
void mergedatafn(int user, int file, int record);
```

Where `record` is the record number or -1 for the next record. Mail merges will start with record==0 or ==1 if the headings record is skipped, after that record will =-1.

The other function prototypes are:

```
void mergefn(int user, int file);
```

Constant Macros

This page lists the pre-defined macros. They may be used in the `type()` function, and are used by ***Ovation Pro*** to implement functions on the **Misc➪Insert** menu.

Examples:

```
type("{SoftHyphen}");
type("Date: {ActiveDate}");
```

`ActiveDate`
Expands to active **Date**.

`ChapterNumber`
Expands to current **Chapter number**.

`Date`
Expands to the current **Date**.

`DateTime`
Expands to the current **Date** and **Time**.

`DeleteB`
Deletes the character before the caret. Note that this macro takes the **PC-style delete** option into account and should be used instead of `{Delete}`.

`DeleteF`
Deletes the character after the caret. Note that this macro takes the **PC-style delete** option into account and should be used instead of `{Delete}`.

`FileName`
Expands to the name of the current file.

`FilePath`
Expands to the full pathname of the current file.

`FileDir`
Expands to the full pathname of the directory containing the current file.

`HardHyphen`
Expands to **Hard hyphen**.

`HardSpace`
Expand to **Hard space**.

`PageNumber`
Expands to current **Page number**.

`NewFrame`
Expands to a new frame.

`NewLine`
Expands to **New line**.

`NewPage`
Expands to **New page**.

`NewPara`
Expands to new paragraph character `(Return)`.

`SoftHypen`
Expand to **Soft hyphen**.

`TabChar`
Expand to the Tab character. Note `{Tab}` is the key macro for the tab key.

Time
Expands to the current **Time**.

emspace
Breaking, non-delimiting em space

emspace_d
Breaking, delimiting em space

emspace_n
Non-breaking, non-delimiting em space

emspace_nd
Non-breaking, delimiting em space

enspace
Breaking, non-delimiting en space

enspace_d
Breaking, delimiting en space

enspace_n
Non-breaking, non-delimiting en space

enspace_nd
Non-breaking, delimiting en space

The following constant macros expand to single character codes. In addition to these any glyph name in the ***Ovation Pro*** glyph list file will be recognised as a constant macro and expand to the relevant character code.

dagger
The † glyph

daggerdouble
The ‡ glyph

ellipsis
The … glyph

emdash
The — glyph

endash
The – glyph

euro
The € glyph

fi
The fi ligature glyph

fl
The fl ligature glyph

guilsingleleft
The ‹ glyph

guilsingleright
The › glyph

minus
The − glyph

oe
The œ ligature glyph.

OE
The Œ ligature glyph.

perthousand
The ‰ glyph

quotedoubleleft
The “ glyph

quotedoubleright
The ” glyph

quotesingleleft
The ‘ glyph

quotesingleright
The ’ glyph

trademark
The ™ glyph

wcircumflex
The ŵ glyph

Wcircumflex
The Ŵ glyph

ycircumflex
The ŷ glyph

Ycircumflex
The Ŷ glyph

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z