# Contents

# Introduction

## Installing WimpWorks

Installing *WimpWorks 2* is simply a matter of copying the files from the program disk to a hard disk or another floppy. For details on how to copy files, create directories and backup disks please refer to the *Welcome Guide* which came with your computer.

Making at least one backup in this way is highly recommended so that if using the program from floppy then if the disk is accidentally corrupted (eg. spilt coffee, young children etc.) then a replacement disk can be made from the original master disk.

If your disk develops a fault please do not hesitate to contact Jaffa Software for a replacement (address at end of manual).

# The WIMP

Acorn's RISC OS computers include a *GUI* (*Graphical User Interface*) or *WIMP environment* which makes using applications and manipulating files much easier. This is called the *Desktop*.

The Desktop is split into three main areas:

- The *icon bar* - the light grey strip along the bottom of the screen which shows device icons on the left (such as the floppy disk drive), and application icons on the right (such as the Task Manager).
- The *pinboard* - the darker grey part of the screen above the icon bar. The pinboard can hold icons and windows in icon form so that they may be accessed quickly when needed.
- The *workarea* - the part of the Desktop in which windows appear. It initially is empty and covers the whole of the Desktop area - this means that windows can be positioned anywhere; over the icon bar, over the pinboard or both.

*WIMP* stands for **W**indows, **I**cons, **M**enus and **P**ointers which are the main elements of any GUI system, such as the Desktop or Microsoft Windows™.

The *pointer* is controlled by moving the mouse and consists of three buttons which are (from left to right):

- *Select* - this is used most of the time to select icons displayed on the screen, or to choose a menu item.
- *Menu* - this button usually makes a menu appear under the pointer.
- *Adjust* - this does a variety of things, mostly useful variants of what the select button does, eg. choosing a menu item and keeping the menu open.

## Windows

Windows are rectangular areas of the screen in which an application can input or output data. Around the window are various controls to allow them to be used more efficiently. Not all windows have to have all the available controls, but a full window would have (clockwise from top-left):

- *Back icon* - puts the window behind all other windows.
- *Close icon* - closes the window.
- *Title bar* - displays the title of the window, and also allows the window to be moved.
- *Toggle size* - switches a window between full size and the last size displayed. (Full size is either large enough to display everything in the window, or such that the window fills the whole screen.)
- *Up arrow* - scrolls the window a fraction upwards, ie. displays the data slightly above that already shown.
- *Vertical scroll bar* - shows which part of the window is in view.
- *Down arrow* - performs the opposite function to the up arrow.
- *Adjust size icon* - lets you alter the size and shape of a window.
- *Horizontal scroll bar* and *left/right arrows* - same in usage as the whole of the vertical scroll bar.

## Icons

Icons are *buttons* in windows (or on the icon bar) which can be configured to do different things, such as to input or display data in a variety of ways, or run BASIC subroutines when clicked. Most things you see in a *WimpWorks* window will be an icon. The full range of icon types are, classified by function follows:

- *Never* - never reports any clicks.
- *Always* - always reports a click whilst the pointer is over the icon.
- *Auto-repeat* - reports multiple clicks when the mouse is held down.
- *Click* - reports a single click when a mouse button is pressed.
- *Release* - reports a single click when a mouse button is released.
- *Double-click* - reports only double clicks on the icon.
- *Click/Drag* - reports both single clicks and drags of the icon.
- *Release/Drag* - similar to Click/Drag.
- *Double/Drag* - a combination of Double Click and Drag.
- *Menu* - reports single clicks, icon inverts when pointer over it.
- *Double/Click/Drag* - reports drags, single and double clicks.

- *Radio* - stays on or off even after the button has been released.
- *Write/Click/Drag* reports clicks and drags and the icon can also be changed by typing in it.
- *Writable* - an icon which can be changed by typing in it.

## Menus

A menu in the Desktop is similar to a menu in restaurant - it's a list of things to choose from, usually related commands and the benefit is that the user does not have to remember obscure key presses to perform these commands. Menus can lead to further *submenus* and these in turn can lead to other submenus recursively. Items can also be greyed out (to make selection impossible) or separated by a dotted line.

Menus are so integral to the Desktop that a mouse button has been set aside for them - the middle mouse button should only be used to create a menu on screen and nothing else. After a menu item has been chosen (using select or menu on the mouse) then the menu is automatically closed, however, when adjust is used the item is chosen and the menu stays on the screen.

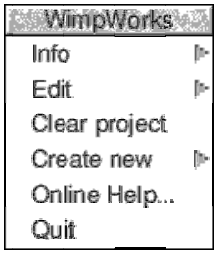
# Summary of WimpWorks' features

*WimpWorks 2* is the latest version of a powerful *integrated development environment* (*IDE*) which allows anyone with a simple knowledge of BBC BASIC V (as fitted to all Acorn RISC OS computers) to produce stunning multitasking applications using windows, icons and menus. Simple applications such as a clock can be created, as can complex systems such as database. Some of the main features are:

- Over 100 new commands in addition to all the standard BASIC ones, such as FOR...NEXT and LEFT$ and MID$.
- Unlimited number of subroutines, menus and windows (upto available memory allows).
- Unlimited length subroutines and menus.
- BASIC's ARM Assembler can also be used to make speed critical parts even faster.
- Expandable system using *WEM*s (*WimpWorks Extension Modules*) and additional tools/editors can be added.
- Full online help for all 110 commands.
- Low memory and disk requirements.
- Support for connections to the World Wide Web.
- Support for *ActiveApps* application communication protocol.
- Unlimited **free** customer support.
- Uses standard BBC BASIC - no obscure new dialect to learn.
- Uses your own text editor, such as *StrongEd*, *Zap* or *Edit*.
- Produced applications are completely stand alone - no runtime modules need to be installed, so your programs can be distributed without any dependency on *WimpWorks*.
- All applications produced with *WimpWorks* can be distributed by you with no royalty fee payable to Jaffa Software - even commercial programs!

## The Icon bar menu

After loading *WimpWorks* by double-clicking on its directory display icon, the icon will appear on the right side of the icon bar. Click menu and the icon bar menu will appear:



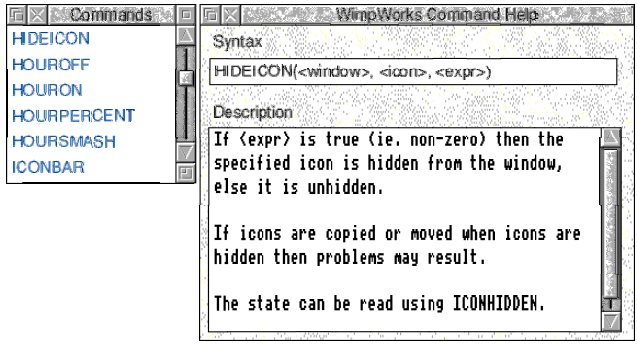

**Info** leads to a standard program information box, although if you have a web browser installed then clicking select on the blue 'Author' text will start your browser and go to a local web page which contains a link to the *WimpWorks* website (at http://www.cryogen.com/jaffa/).

*WimpWorks* is based around a system of editors which control the various aspects of your program, the default editors are the *Task*, *Window*, *Menu* and *Subroutine* editors. **Edit** is a submenu of all the valid editors in !WimpWorks.Resources.Editors and is greyed out until an application is loaded.

**Clear project** simply removes the current application from memory, if it has not been saved then you will be asked if you wish to **Discard** the changes, **Cancel** the operation or **Save** the application.

**Create new** will be explained below.

**Online Help...**, choosing this will open a window listing all the commands available, including any provided by a *WEM* (*WimpWorks Extension Module*). Clicking select on a command will show a full description of that command, and clicking adjust on the command will automatically type the command into whatever window currently has the *input focus* (ie. the current text cursor).

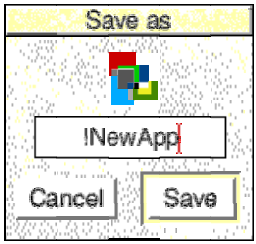


# Creating/Editing an application

## Creating, loading, saving and information

### Creating a new application

To create a new application, click select on the icon bar icon, the **Create New** box will appear:




By dragging this to a directory display a new, blank application will be created and automatically loaded into the editor.

### Loading an application

As stated above, when an application is created it is automatically loaded - however to load an old application then drag the application from a directory display to the *WimpWorks* icon.

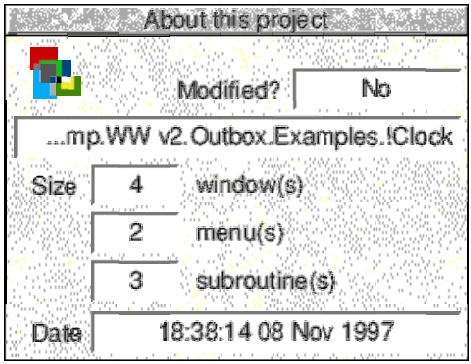
### Saving the application

To save the application then either click **Save** on the **Project** menu (click menu over the workarea of either the task, menu or subroutine editor), alternatively press Ctrl-F3 when any of those editors has the input focus.

After a short time the application will have been saved, and it is then completely ready to run - it is stand alone so no runtime modules need to be installed.

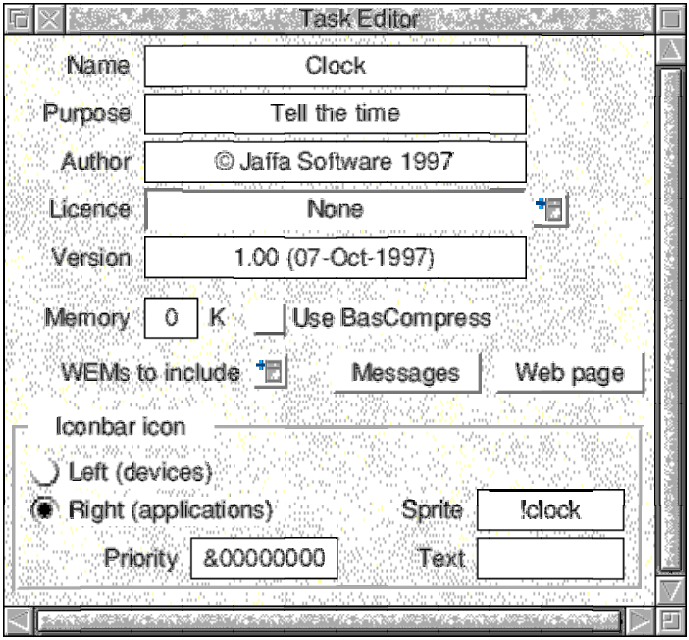
## Information on an application

When the input focus is in any of the *central* editors (task, menu or subroutine - the window editor is separate, see later) then pressing Ctrl-F1 or choosing **Info** from the **Project** menu will display information on the current application:



# The Task Editor

Choosing **Task details** from the **Edit** menu will open the *Task Editor* - this is also opened as standard when a task is loaded.




## The Information Box

The top five lines show the information which appears in the application's info box - eg. off the **Info** item in the iconbar menu.

The format of the version is usually 'x.xx (dd-mmm-yyyy)' and the '©' symbol can be obtained by pressing Shift, Alt and C at the same time.

The licence type is optional and if it is equal to 'None' then no licence field is shown in the information box. The popup menu contains several common types, but the bottom entry is editable for your own types.

## Memory

This field allows you to specify an extra amount of memory to be allocated to your task - usually memory allocation is dynamic and automatic. However, if your program includes a large **DIM** then this memory slot will need to be increased. For this reason **CLAIM** (see later) should be used for memory allocation as much as possible.

If memory is too low then strange errors may occur, or certain features of your program may simply not work - this error can be difficult to track down.

## Use BasCompress

BasCompress is a BASIC Compressor which takes BASIC code and compresses the spaces, removes the comments and compresses variable names. If BasCompress has been seen by the Filer and this option is selected then your code will be run through BasCompress before being linked with the core library.

This option also disables the *Debug* button which appears on the error box when an error in your code occurs and WimpWorks is loaded. Therefore if BasCompress is not present and this option is ticked then no error will be produced: the *Debug* disable will still be useful if, for example, you were producing a commercial application.

## WEMs to include

WEMs (*WimpWorks Extension Modules*) are software components which allow new commands and events to be incorporated into your WimpWorks programs. WEMs are installed by copying them into !WimpWorks.Resources.WEMs and this menu provides a list of which WEMs should be included in your code and which should not.

No WEMs are included, by default, in a new application.

## Messages/Web page

These two buttons edit the *Messages* file for your application (note that the first two messages **must** be present) and the *Web page* for your application respectively.

The Messages file allows the text for your application to be stored in one place, making internationalisation much easier as only one file needs to be translated. The format of the messages file is a list of tokens (which can be looked up using TOKEN and SUBST), eg:

```
#A comment
Token1:The text for this token
Token2:%0 is replaced if using SUBST as is %3
```

The web page is copied into your application as *HomePage* and if present, and a web browser is installed on the system then the *author* line in the information box will be blue (and the pointer will change to a hand over the text); clicking on this will start the web browser with your web page loaded.

The web page could contain a  link to your Internet homepage, your email address or just a more graphically appealing help file.

## Icon bar information

By default, each application will have one icon on the icon bar (see later for other possibilities), this portion of the task editor allows you to specify how it will appear - most of them are self explanatory, except for the **Priority** field.

This field allows you to specify whereabouts on the icon bar your icon will appear - it is most useful for device icons and filing systems which should be grouped according to their type.

Example priorities are:

| **Module/Task** | **Priority** |
|---|---|
| Task Manager | &60000000 |
| !Help | &40000000 |
| Palette Utility | &20000000 |
| Applications | 0 |
| | |
| ADFS hard disks | &70000000 |
| ADFS floppy disks | &60000000 |
| ‘Apps’ icon | &50000000 |
| RAM disk | &40000000 |
| Ethernet | &30000000 |
| Econet | &20000000 |
| Other filing systems | &10000000 |
| Printer drivers | &0F000000 |
| TinyDirs | &0E000000 |

# The Window Editor

The *Window Editor* is available by choosing **Windows** from the **Edit** menu. To begin with the *Window Display* is opened:

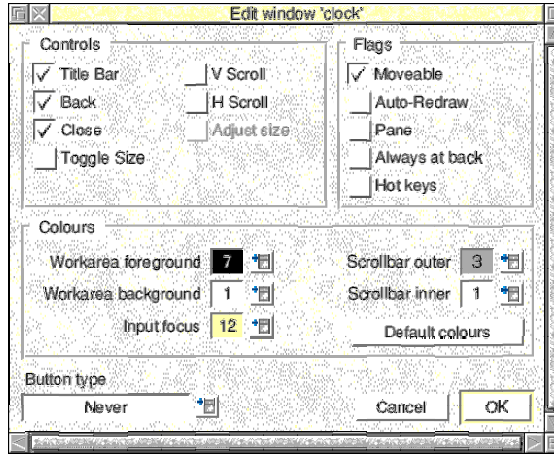

Note that the first three windows, *error*, *saveas* and *info* are required if your program is to have an extended error box, save box or program information box respectively. However, your program will still work without any of them.

There is a menu available by clicking menu on this window which allows you to copy, rename and delete windows. You can also choose **Save** from this menu to save the windows, before this your application's windows are *not* updated. Closing this window will also allow you to save them back to your application.

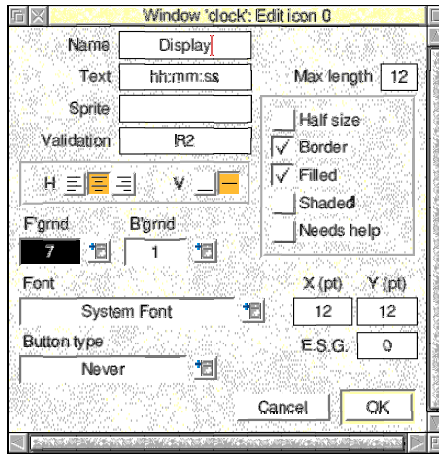
By double clicking on a window name or icon then a copy of the window will be opened (as with the clock window above). The version of this window which appears in your application will initially be in the same position, the same size and so on.

By clicking menu on a window then a menu will appear which allows you to change the maximum and minimum size of the window, edit the title or edit the properties of the window (**Edit window...** or by clicking menu with Ctrl held down). In both the *window* editor and the *icon* editor then certain keys have meanings:

| | |
|---|---|
| **Return** or **F5** | Next field or close window and save changes |
| **Escape** or **F4** | Discard changes and close window |




Icons can be moved by dragging them with select and resized by dragging with adjust, they can also be copied and created by clicking menu over a window. The **Create icon** submenu has a list of standard icons which can be edited to suit your own needs. This can be done by choosing **Edit...** from the **Icon** menu or by clicking menu over the icon with Shift held down:




For a description of the button types see the introduction to the Wimp earlier.

For people who have used template editors before this window may be a bit confusing as it is *a lot* simpler than most icon editors. Several points to note are:

- The **Name** field - when the application is run the variable *window_name* is set to the icon number specified here. So if the window is ‘clock’ and the name of the icon is ‘Display’ then clock_Display is set to, for example, 0. The name should only contain alphanumeric characters and ‘_’.
- There are no options for choosing whether or not there is text and/or a sprite present; these are deduced by whether or not there are values in the relevant fields. If there is text and a sprite then the sprite data is automatically placed into the validation string (and extracted again when re-editing the icon).
- When using outline fonts the colours do not have to be placed in the validation string as, eg. F17, the colour selectors can be used as normal.
- The maximum size field limits the maximum number of characters which can be placed in the icon. By clicking select on **Max. Length** itself then the field is reduced (or increased) to the number of characters necessary to handle the data in the text and sprite fields.
- **Needs help** means that the icon should be fully redrawn by the Wimp, it can usually be left off.

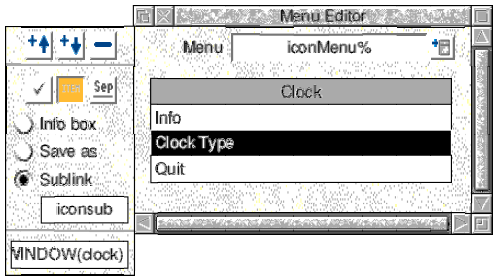
## Exclusive Selection Group (E.S.G.)

The ESG of an icon indicates to which group an icon belongs, there are a maximum of 32 groups, including 0. Only one icon in a group may be selected at a time (groups are unique to each window) except group 0 which is the default and may have many icons selected at any one time.

The main use of ESGs is for groups of radio icons such as the horizontal and vertical position icons in the icon editor.

# The Menu Editor

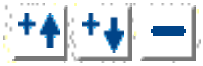
Choosing **Menus** from the **Edit** menu will open the menu editor at the first menu (if one is defined):




The editor is split into two parts, the main window and the toolbox to the left.

## The toolbox

Taking the toolbox first, the top three icons are:

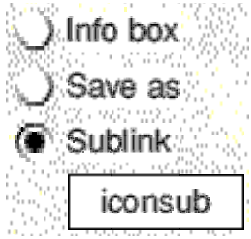


The first icon inserts a new blank item *above* the currently selected item, the second inserts a new item *below* the current item and the final icon deletes the current item.

The next three icons:


These show the flags of the item, they are ticked, greyed out and followed by a separator bar, respectively.

The next set of icons describes what the item links to:




**Info box** means that the item will link to a standard program information box as defined in the *Task Editor*.

**Save as** will lead off to a standard 'save as' box. The icon and text of which are those last set with **SETSAVE**.

**Sublink** means just a general link to another menu or window, the identifier of which should be typed into the writable icon.

The last writable field in the toolbox holds the BASIC command which should be executed when the item is chosen. This can be something simple, such as VDU 7 or something more complex such as:

```
IF RND(2)=1 THEN OPENWINDOW(clock)
```

Note that this field can also contain WimpWorks commands (as above).

## The main window

The **Menu** field shows which menu is currently being edited, you can choose between the defined menus using the popup menu icon. The bottom item is **New menu** which will allow you to create a new menu.

By clicking menu on the **Menu** field when a menu is being edited then you have the option to copy, rename and delete menus. Although you can copy a menu to a name which is already taken this is definitely not recommended when your program is run as memory will be wasted and only one of the menus will be properly defined.

Below this is a representation of the menu itself, the darker grey bar is the title bar of the menu and below this are the items themselves - by default there are no items defined, however, there is *always* a titlebar. Items can be created by clicking on the *insert below* icon in the toolbox whilst the titlebar is selected.

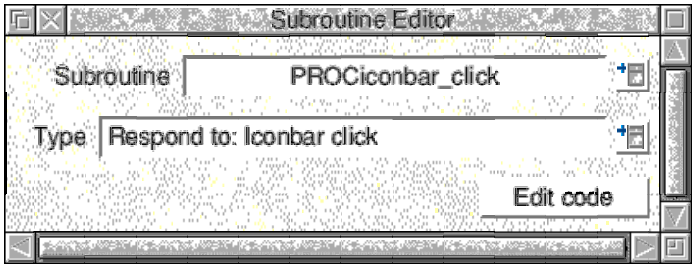
To edit an item (or the titlebar) click on it once, this will highlight the item. Change the text and flags etc. as required, then either click on one of the *insert* icons in the toolbox, click on a different item or click on the same item again - this will save the changes you have made.

Forgetting this last step, ie. changing menus or closing the menu editor, will mean that the changes to that item are also forgotten. This can be useful when you realise that you did not want to change the item in that way.

For details on how to create a writable menu item see the *Hints & Tips* section later in this manual.

# The Subroutine Editor

The *Subroutine Editor* is where you define what happens when certain events happen within your application and is where you write the actual code for your application. Choosing **Subroutines** from the **Edit** menu opens the window:




## Using subroutines

The subroutine editor is very similar to the menu editor in operation: subroutines can be copied, renamed and deleted by clicking menu over the **Subroutine** field, and new subroutines or old ones can be chosen by clicking the top pop-up icon. The actual code can be edited by clicking on **Edit code** - this opens the subroutine in the current text editor, normal BASIC can be typed here, as can any of the new commands (see later), such as **OPENWINDOW**.

## Subroutine Types

Once you have selected a subroutine to edit you then have to choose its type; this is done by clicking select on the lower pop-up icon. The possible types are:

- *Normal* - this is like an ordinary PROC or FN in BBC BASIC, it will only be called when you specifically execute PROC*name* or FN*name*.

- *Call Every* - this type of subroutine (again, PROC or FN) will be called at regular intervals. The length of time (in centiseconds) in specified in the submenu off the **Call Every** menu item.
- *Respond to event* - these can either be PROCs or FNs depending on whether or not a value is expected to be returned. You will only be able to select an event if it matches with PROC or FN.

All subroutines can of course still be called with PROC*name* or FN*name*, however the latter two types will also be invoked by the system.

## Events

Events are called when certain things occur to your program which require you to determine what happens next, examples are when an icon is clicked, *Interactive Help* is used on your program or when the user wants to quit. There are 19 events by default, however others may be provided by a WEM. The full list is:

**Starting Up**

*Description*: Called when your task is started, this is guaranteed to be called only once during the time when your task is active.

*Inputs*: None, but can access **argc%** and **argv$()** which are a count of the command line options and an array of them respectively

*Outputs*: None

**Null**

*Description*: Called when the WIMP is idling, however this is only called if **BUSYON** has been called.

*Inputs*: **time%** - the number of centiseconds since the last call

*Outputs*: None

**Closing down (FN)**

*Description*: Called when the **CLOSEDOWN** command has been executed.

*Inputs*: None

*Outputs*: Return **TRUE** to exit or **FALSE** to continue

**Iconbar click**

*Description*: When the user clicks on the icon bar icon (the one defined in the task editor).

*Inputs*: **button%** - the mouse button used

*Outputs*: None

**File dragged**

*Description*: Called when an object from a directory display is dragged to an object belonging to your task.

*Inputs*: **window%** - the window it was dragged to (-1 if iconbar), **icon%** - the icon the pointer is over (-1 if none), **file$** - the name of the object, **type%** - the object's file type (&000-&FFF, &1000 or &2000)

*Outputs*: None

**File double-clicked**

*Description*: Called when an object is run from inside a directory viewer.

*Inputs*: **file$** - the name of the object, **type%** - the object's file type

Outputs: None, use **LOADACK** to stop other tasks trying to load it

**Help request (FN)**

*Description*: Should return a string for the *Interactive Help* program.

*Inputs*: **window%** - the window the pointer is over (-1 if iconbar), **icon%** - the icon the pointer is over (-1 if none)

*Outputs*: Return the string to be displayed

**Key pressed (FN)**

*Description*: Determines whether or not a keypress should be passed on to further tasks.

*Inputs*: **window%** - the window handle in which it occurred, **icon%** - the icon handle in which the key was pressed, **key%** - the key code of the key pressed

*Outputs*: Return **TRUE** if the key should be pressed on to other tasks (eg. F12), or **FALSE** if you have claimed it yourself

**Data saved (FN)**

*Description*: Called when a drag from a save box has been completed.

*Inputs*: **file$** - the file to which data should be saved

*Outputs*: Return **TRUE** if you managed to save it without any errors, otherwise return **FALSE**

**Menu selected**

*Description*: Called whenever a menu item is chosen, useful when using dynamic menus.

*Inputs*: **menu%** - the menu handle of the item, **item%** - the item number (0 for first, at top of menu), **text$** - the textual contents of the menu item (eg. for use in writable menus)

*Outputs*: None

**Window opening (FN)**

*Description*: Informs the system as to whether or not a specific window can be opened.

*Inputs*: **window%** - the window wanting to be opened

*Outputs*: Return **TRUE** if the window is allowed to open, else return **FALSE**

**Window closing (FN)**

*Description*: Informs the system as to whether a specific window can be closed.

*Inputs*: **window%** - the window wanting to be closed

*Outputs*: Return **TRUE** if the window can close, else return **FALSE**

**Window clicked**

*Description*: Called when an object in a window has been clicked

*Inputs*: **window%** - the window in which the click occurred, **icon%** - the icon handle, **button%** - the mouse button used to click the object

*Outputs*: None

**ActiveApps command**

*Description*: Called when your program receives an *ActiveApps* command from another task.

*Inputs*: **task$** - the name of the task which sent the command, **reference%** - the unique reference generated by **COMMAND**, **command$** - the contents of the command itself

*Outputs*: None

**ActiveApps reply**

*Description*: Called when your task receives an *ActiveApps* reply.

*Inputs*: **task$** - the name of the task which sent the reply, **reference%** - the unique reference passed in the command, **reply$** - the contents of the reply itself

*Outputs*: None

**Pointer entering window / Pointer leaving window**

*Description*: Called when the mouse pointer enters (or leaves) the area of one of your windows.

*Inputs*: **window%** - the window handle

*Outputs*: None

**Slider changed**

*Description*: Called when one of the sliders defined using **MAKESLIDER** is dragged to a new value.

*Inputs*: **window%** - the window which contains the slider, **icon%** - the icon number of the slider background, **value%** - the new slider value, 0 - 100

*Outputs*: None

**POLL called**

*Description*: Called when the system has completed its POLL loop. Note, this is recommended to advanced users only.

*Inputs*: **block%** - a pointer to the 256 byte block used in Wimp_Poll, **action%** - the action code returned from Wimp_Poll

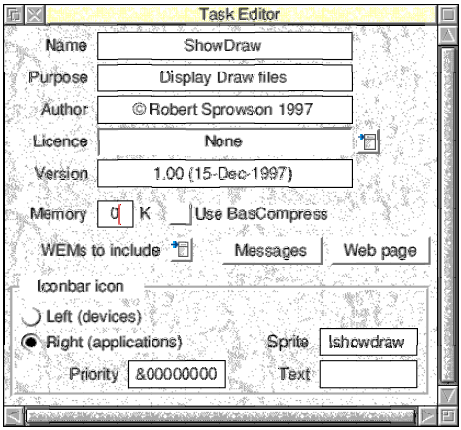
*Outputs*: None

# Tutorials

## Example 1 (Simple)

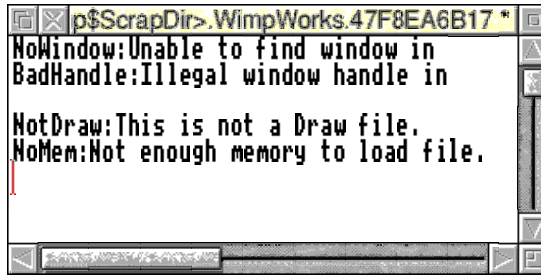
This example is a step-by-step walk through of *ShowDraw*. Start *WimpWorks 2* in the normal way. After the start up banner has disappeared and the icon is on the icon bar, click menu on this icon and go to the **Create New** window. Change the filename to *!ShowDraw* before dragging the sprite to a directory display.

The *Task Editor* window opens, where you can fill in details that will end up in the info box:

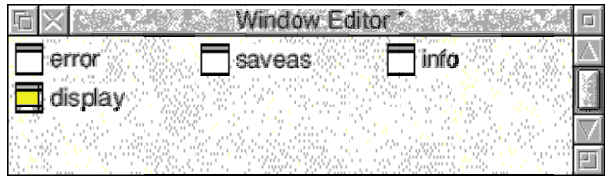



Click on the popup icon for **WEMs to Include** and choose **Starter Pack WEM**, ensuring that a tick is now next to it.

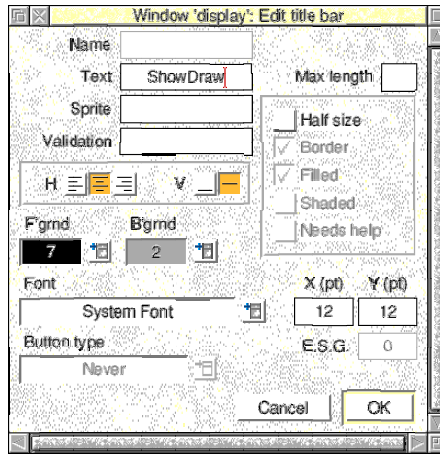
Pressing the **Messages** button opens a text file in your chosen editor (eg. *Edit*). After the two messages already present, add the following messages, then save these changes and close the editor's window.



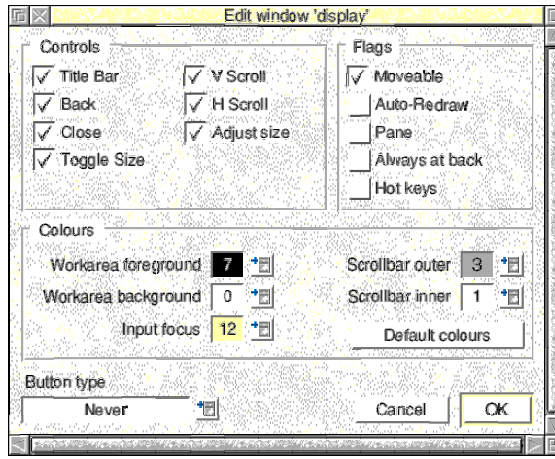

Moving back to the icon bar icon choose **Windows** on the **Edit** submenu, this loads the *Window Editor*. Click in the window that opens and **Create** a new window called *display*. A new window should open and the *Window Editor* window should now look like:



Click menu on the titlebar of the new window (which should say *display*) with shift held down. This will open the *Edit title bar* window, change the title to *ShowDraw* and click on **OK** or press **F5**.

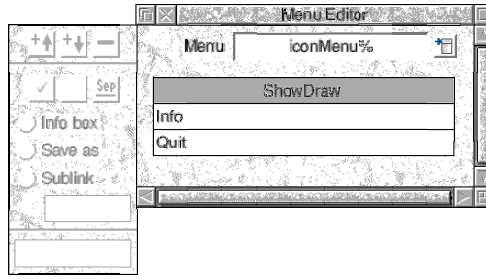



Now click menu on the workarea (the large grey area of the window) with ctrl held down. This will open the *Edit workarea* window, change the workarea colour to white and click on **OK** or press **F5**.



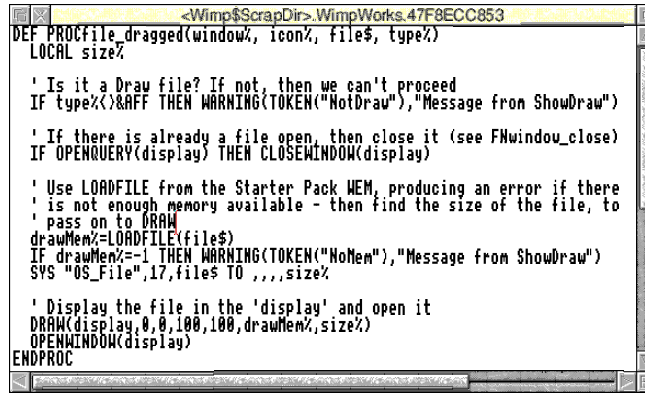

Close the *Window Editor* and choose to **Save** your changes.

Moving back to the icon bar icon, choose **Menus** on the **Edit** submenu. The *Menu Editor* will open, however the menu shown, which will open on a menu click on the iconbar, is all we need so this window can be closed without any changes.



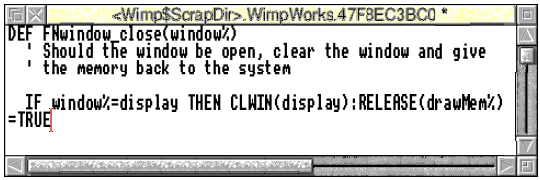

Moving back to the icon bar icon, choose **Subroutines** on the **Edit** submenu. After the *Subroutine Editor* has opened you need to create a new subroutine. Click on the popup icon next to the **Subroutine** field and enter *PROCfile_dragged* into the **New Subroutine** box. Change the type of the subroutine by clicking on the lower popup icon and make it **Respond to** the **File dragged** event.

Now click on **Edit code** and change the window which opens in your editor to read:




Note that **LOADFILE** is provided by the *Starter Pack WEM* which we included earlier. Save your changes and close the window.

Create a second subroutine, as before, but call it *FNwindow_close*. This should respond to the **Window closing** event. The code for this function is:




Save your changes and close the window as before.

Press menu on the background of any of the *WimpWorks* editors open and choose **Save** from the **Project** menu.

You have now written your first *WimpWorks* program **and** it is ready to run! Double click on its icon and drop a Draw file onto its icon bar icon.

A window should open showing its contents, if an error does occur and *WimpWorks* is running, pressing the **Debug** button will take you to the point in the *Subroutine Editor* at which the error occurred. Note, however, that this feature requires that your editor supports **F5** to go to a specific line.

This simple example could be expanded to allow the zoom and ratio to be altered etc. This is left as an exercise for the reader.

# Example 2 (Complex)

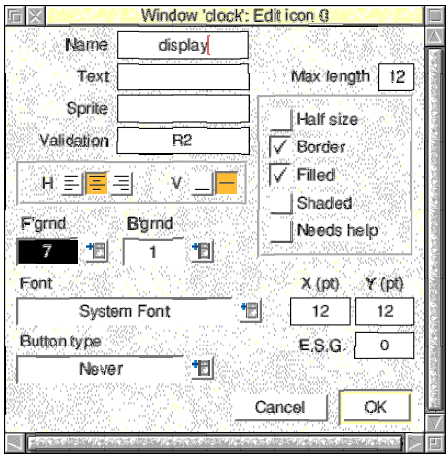
This is a quicker run through of a more advanced example, *Clock*.

Firstly, create the application, *!Clock* by using **Create New** as above. The task details are not really important in this example and no messages need to be defined, or WEMs included. Only the information box needs information, and something like the following is suggested:

```
Name        Clock
Purpose     Tell the time
Author      © Your name 1998
Version     1.00 (dd-mmm-yyyy)
```

Then open the *Window Editor* (choose **Windows** on the **Edit** submenu), and create a new window called *clock*. Change the title of this window to *Clock* and drag the adjust size icon with ctrl and alt held down to shrink the maximum size of the window to a more comfortable size.

Click menu on the workarea of the window about ¾ of the way down, choose **Display** on the **Create icon** submenu. A new icon will appear in the window. Drag it into position using select and then edit its attributes by clicking select on it with shift held down.



Click **OK** or press **F5**, close the *Window Editor*, choosing to **Save** your changes.

Open the *Menu Editor* (by choosing **Menus** on the **Edit** submenu) and click on *Info* (in the *iconMenu%* menu) highlighting it.

Then click on the 'insert after' button (which shows ↓+); a new, blank item is inserted between *Info* and *Quit* and is automatically highlighted. Enter `Clock Type` as the text and, in the toolbox, set it as a **Sublink** and enter `iconsub` in the field under the radio icon. Click once on this item in the main window to save the changes.

We then need to define the submenu *iconsub* which the new item will link to. Click on the popup icon next to the **Menu** field and under **New menu** enter `iconsub` (remembering to press return).

The new menu is now shown and can be seen to be blank. Click on the box representing the title bar of this menu and change the text to `Clock Type`, then click on the 'insert after' button again. The new item becomes highlighted, set the text to be `HH:MM:SS`. Ensure that the toolbox shows that this item is ticked, and then in the lowest writable field in the toolbox enter `PROCtype_click(0)` - this is the command which will be executed when that item is clicked.

Click on 'insert after' one last time, the new item should have text `HH:MM` and the command should be `PROCtype_click(1)`. Click once on the item again to save the changes and then open the *Subroutine Editor* (**Subroutines** on the **Edit** submenu).

We need to define three subroutines, the first, *PROCtype_click* has just a **Normal** type, and the code as follows:

```
DEF PROCtype_click(item%)
   LOCAL index%

   FOR index%=0 TO 1
      CHANGEITEM(iconsub, index%, "", " ")
   NEXT
   CHANGEITEM(iconsub, item%, "", "Ticked")
ENDPROC
```

The second subroutine is called *PROCupdate_clock*, this need to be called every second, so under the **Call Every** item on the **Type** menu enter 100. The code for *PROCupdate_clock* is:

```
DEF PROCupdate_clock
   LOCAL time$

   time$=SYSTIME
   IF ITEMTICKED(iconsub,1) THEN
      time$=LEFT$(time$,5)
   ENDIF
   SETICON(clock, clock_display, time$)
ENDPROC
```

Finally, we need to give the user a way of seeing the clock and so we define *PROCiconbar_click* which will **Respond to** an **Iconbar clicked** event:

```
DEF PROCiconbar_click(button%)
   ‘ If select pressed open the window

   IF button%=4 THEN OPENWINDOW(clock)
ENDPROC
```

Save the application and run it, using the *Debug* button if necessary to correct any mistakes.

# Command Reference

This section contains details on the new *WimpWorks* commands which can be used alongside the standard BBC BASIC ones in your programs.

Each entry consists of:

## THE COMMAND

**Syntax** `THE COMMAND(`$p_1$`, `$p_2$`, ..., `$p_n$`)`

**Use** A description of the what the command does.

**Example** A simple example of the command in use

**See also** A list of any commands which are related to this.

# ADDITEM

**Syntax** `ADDITEM(`*`menu, text, control`*`)`

**Use** A new menu item with text as contained in the string *text* is added to *menu* (cannot be the icon bar menu), the *control* string describes the flags for the item:

| | |
|---|---|
| `T` | Ticked |
| `L` | Lined off (separator) |
| `G` | Greyed out |
| `S(`*`link`*`)` | Sublink to *link* |
| `W(`*`addr, len`*`)` | Writable (memory at *addr*, max length in *len*) |

**Example**
```
ADDITEM(mymenu%,“Submenu 1”,“S(submenu1%)”)
```

**See also** CHANGEITEM, NEWMENU

# AFTER

**Syntax** `AFTER(`*`time, subroutine`*`)`

**Use** Calls *subroutine* after *time* centiseconds, once and once only.

**Example**
```
AFTER(100,“PROCbeep_after_1s”)
```

**See also** EVERY

# BUSYOFF

**Syntax** `BUSYOFF`

**Use** This prevents *Null* events from occurring which is the default - hence this will do nothing unless null events have been enabled using BUSYON.

**Example** `BUSYOFF`

**See also** BUSYON, POLL

# BUSYON

**Syntax** `BUSYON`

**Use** Enables *Null* events, events which occur when nothing is happening in the Desktop.

**Example** `BUSYON`

**See also** BUSYOFF, POLL

# CEILING

**Syntax** `CEILING(`*`expr`*`)`

**Use** Returns the ceiling of *expr*, ie. the next highest integer if *expr* is not exactly an integer. Hence CEILING rounds up, whereas INT (supplied with BASIC) rounds down.

**Example**
```
four%=CEILING(3.2)
```

**See also** INT, ROUND

# CENTREWIN

**Syntax** `CENTREWIN(`*`window`*`)`

**Use** Centres *window* on the screen and then opens it on the top of the window stack.

**Example**
```
CENTREWIN(dialogue)
```

**See also** OPENWINDOW, MOVEWINDOW

# CHANGEITEM

**Syntax** `CHANGEITEM(`*`menu, item, text, control`*`)`

**Use** Changes *item* (0 is the top item) in *menu* to contain the new *text* and with the flags as set in *control*. If *text* or *control* are empty strings, ie. ““, then that part of the menu is not changed.

A separator (the *L* flag) cannot be removed using this command, but can be added.

**Example**
```
CHANGEITEM(iconMenu%,0,“”,“G”)
```

**See also** ADDITEM

# CIRCLE

**Syntax** `CIRCLE(`*`window, X, Y, radius, colour`*`)`

**Use** Draws a circle outline at *X*, *Y* in *window* with radius *radius*. All units are OS units ($^{1}/_{180}$ th of an inch - depending on monitor size). *colour* is a 24-bit colour of the form: &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
CIRCLE(main_window,200,-200,300,&0000FF00)
```

**See also** CIRCLEFILL, CLWIN

# CIRCLEFILL

**Syntax** `CIRCLEFILL(`*`window`*`, `*`X`*`, `*`Y`*`, `*`radius`*`, `*`colour`*`)`

**Use** Draws a filled circle at *X*, *Y* in *window* with radius *radius*. All units are OS units ($^{1}/_{180}$ th of an inch - depending on monitor size). *colour* is a 24-bit colour of the form: &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
CIRCLEFILL(main_window,200,-200,300,&0000FF00)
```

**See also** CIRCLE, CLWIN

# CLAIM

**Syntax** `CLAIM(`*`expr`*`)`

**Use** Returns a pointer to a block of memory at the top of the application workspace, of size *expr* bytes. Hence it is similar to DIM (as supplied in BASIC), however it is much more useful as if the memory cannot be allocated, -1 will be returned. Also, by using CLAIM the amount of memory your task uses can vary. It is strongly recommended that CLAIM be used to reserve memory instead of DIM.

**Example**
```
pointer%=CLAIM(65536)
```

**See also** CLAIMMORE, DIM, HEAPSIZE, RELEASE

# CLAIMMORE

**Syntax** `CLAIMMORE(`*`pointer, expr`*`)`

**Use** Extends a the amount of memory in *pointer* by *expr* bytes (assuming *pointer* was reserved using CLAIM). If this operation could not be performed then *pointer* will not change, however it **may** change if it is successful.

**Example**
```
CLAIMMORE(pointer%,&100)
```

**See also** CLAIM, DIM, HEAPSIZE, RELEASE

# CLEARALL

**Syntax** `CLEARALL(window, expr)`

**Use** Sets all the writable icons in *window* to be filled with the string *expr*. If *window* does not contain any writable icons then no changes will be made to the window.

**Example**
```
CLEARALL(mainWin,“42”)
```

**See also** SETICON

# CLOSEDOWN

**Syntax** `CLOSEDOWN`

**Use** Quits the application if the *Closing down* event is undefined or returns true.

**Example** `CLOSEDOWN`

**See also** -

# CLOSEMSGBOX

**Syntax** `CLOSEMSGBOX(`*`expr`*`)`

**Use** Closes the extended message box, if open, and returns *expr* as the number of the button pressed.

**Example** `CLOSEMSGBOX(1)`

**See also** MESSAGE, SURE, WARNING

# CLOSESAVE

**Syntax** `CLOSESAVE`

**Use** Closes the ‘save as’ box if open, otherwise does nothing.

**Example** `CLOSESAVE`

**See also** OPENSAVE

# CLOSEWINDOW

**Syntax** `CLOSEWINDOW(`*`window`*`)`

**Use** Removes *window* from the screen if the *Window closing* event is undefined or returns true.

**Example** `CLOSEWINDOW(mainWin)`

**See also** OPENWINDOW

# CLWIN

## Syntax

`CLWIN(`*`window`*`)`

## Use

Clears all the graphics (**not** icons) from *window* and refreshes the screen if necessary. Graphics are drawn using commands such as LINE, DRAW and RECTANGLE.

## Example

```
CLWIN(mainWin)
```

## See also

REDRAW

# COMMAND

## Syntax

`COMMAND(`*`task, command`*`)`

## Use

ActiveApps is an inter-application communication protocol, allowing tasks on one machine, or across the world to talk to each other very simply.

*task* is a string containing the name of the task (as it appears in the *Task Manager*) for whom the message is targeted, or an empty string ("") if it should be broadcast to all tasks (only on the local machine).

All ActiveApps Aware applications will respond to the following commands, "**Ping**" - the task will reply with "Pong"; "**Quit**" - equivalent to selecting **Quit** from the Task Manager; "**KILL**" - the task will exit immediately, performing minimal tidying up.

This command will return a unique reference integer which will be passed if the task replies to this.

## Example

```
pingRef%=COMMAND("WimpWorks","Load
ADFS::Disk1.$.!App")
```

## See also

EXECCMD, TASKHANDLE, TASKNAME

# COPYICON

**Syntax** `COPYICON(`*`window`*`, `*`icon`*`, `*`X`*`, `*`Y`*`)`

**Use** Copies the specified icon to *X*, *Y* (top-left) in the same window and returns the new icon number of the icon.

**Example**
```
newOK%=COPYICON(mainWin, mainWin_OK, 0, -200)
```

**See also** DELETEICON, MOVEICON

# DELETEICON

**Syntax** `DELETEICON(`*`window`*`, `*`icon`*`)`

**Use** Deletes the specified icon and redraws the window if necessary.

**Example**
```
DELETEICON(mainWin, newOK%)
```

**See also** COPYICON, MOVEICON

# DELETEMENU

**Syntax** `DELETEMENU(`*`menu`*`)`

**Use** Deletes *menu* and returns the memory used back to the system.

**Example**
```
DELETEMENU(mainWinMenu%)
```

**See also** NEWMENU

# DRAW

**Syntax** `DRAW(`*`window`*`, `*`X`*`, `*`Y`*`, `*`ratio`*`, `*`scale`*`, `*`mem`*`, `*`size`*`)`

**Use** Places the Draw file of *size* bytes, located at the address held in *mem*, into *window* at *X*, *Y* (specifying the top-left). DRAW can be used many times for the same *mem* and *size* to give multiple copies of the same drawing. *ratio* is the horizontal size: vertical size ratio, hence if *ratio* = 50 (ie. 50%) then the picture will be squashed to half size in the horizontal direction. *scale* is the percentage size of the picture (ie. 100% is original size).

**Example**
```
DRAW(mainWin,0,-100,75,50,drawptr%,drawlen%)
```

**See also** CLWIN

# ELLIPSE

**Syntax** `ELLIPSE(`*`window`*`, `*`X`*`, `*`Y`*`, `*`major`*`, `*`minor`*`, `*`angle`*`, `*`colour`*`)`

**Use** Draws an ellipse outline in *window* with the top-left of the bounding box at *X*, *Y* and the lengths of the major and minor axes in *major* and *minor* respectively. *angle* is the angle of the ellipse from the horizontal. *colour* is the 24-bit colour of the ellipse, of the form &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
ELLIPSE(mainWin,0,0,200,100,45,&FF00FF00)
```

**See also** CLWIN, ELLIPSEFILL

# ELLIPSEFILL

**Syntax** `ELLIPSEFILL(`*`window`*`, `*`X`*`, `*`Y`*`, `*`major`*`, `*`minor`*`, `*`angle`*`, `*`colour`*`)`

**Use** Draws a filled ellipse in *window* with the top-left of the bounding box at *X*, *Y* and the lengths of the major and minor axes in *major* and *minor* respectively. *angle* is the angle of the ellipse from the horizontal. *colour* is the 24-bit colour of the ellipse, of the form &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
ELLIPSEFILL(mainWin,0,0,200,100,45,&FF00FF00)
```

**See also** CLWIN, ELLIPSE

# ENCODE

**Syntax** `ENCODE(`*`expr`*`)`

**Use** Returns the string *expr* encoded into a non-unique integer. Although the number is non-unique the values are well distributed and clashes will be rare when applied to most strings.

**Example**
```
result%=ENCODE(input$)
```

**See also** -

# ENDDRAG

**Syntax** `ENDDRAG`

**Use** Ends the current drag operation, if one is in progress, otherwise does nothing.

**Example**
```
ENDDRAG
```

**See also** STARTDRAG

# EVERY

**Syntax** `EVERY(`*`expr, call`*`)`

**Use** Adds *call* to the list of regularly called subroutines. *call* is called every *expr* centiseconds (if possible, or as soon as possible afterwards if not).

**Example**
```
EVERY(100,“PROCupdate_clock”)
```

**See also** AFTER

# EXECCMD

**Syntax** `EXECCMD(`*`expr`*`)`

**Use** Executes *expr* (a tokenised Basic string, or a PROC/FN call).

**Examples**
```
EXECCMD(”global_var%=42”)
EXECCMD(”PROCcall”)
```

**See also** -

# FULLNAME

**Syntax** `FULLNAME(`*`expr`*`)`

**Use** Returns the canonicalised (full path) name of the file passed in *expr*.

**Example**
```
fullpath$=FULLNAME("&.File"): REM fullpath$
could be ADFS::Disk1.$.File
```

**See also** LEAFNAME

# GETCARET

**Syntax** `GETCARET(`*`window`*`, `*`icon`*`, `*`position`*`)`

**Use** Fills *window*, *icon* and *position* with the values relevant to the current position of the caret, ie. where text will appear when typed.

*window* will be -1 if the caret is not visible. *position* is the position of the caret within the icon's string.

**Example**
```
GETCARET(window%, icon%, position%)
```

**See also** SETCARET

# GROUPSTATE

**Syntax** `GROUPSTATE(`*`window, esg`*`)`

**Use** Returns the icon handle of the first icon selected in the specified ESG (Exclusive Selection Group), which are used to produce, for example, radio icons. -1 is returned if no icon is selected.

**Example**
```
option%=GROUPSTATE(mainWin, 1)
```

**See also** READSTATE, SETSTATE

# HEAPSIZE

**Syntax** `HEAPSIZE(`*`addr`*`)`

**Use** Returns the size of memory in the heap block at *addr*, This is the amount allocated by CLAIM and, possibly, changed using CLAIMMORE.

**Example**
```
blksize%=HEAPSIZE(ptr%)
```

**See also** CLAIM, CLAIMMORE

# HIDEICON

**Syntax** `HIDEICON(`*`window`*`, `*`icon`*`, `*`expr`*`)`

**Use** If *expr* is true (ie. non-zero) then *icon* is hidden from the window, else it is unhidden. The display is updated if necessary.

**Example**
```
HIDEICON(mainWin, 1, TRUE)
```

**See also** DELETEICON, ICONHIDDEN, SHADEICON

# HOUROFF

**Syntax**
```
HOUROFF
```

**Use** Decreases the number of hourglasses switched on by one, if the number reaches zero then the pointer is restored to normal.

**Example**
```
HOUROFF
```

**See also** HOURON, HOURSMASH

# HOURON

**Syntax** `HOURON`

**Use** Increases the number of hourglasses currently switched on (only one is ever shown).

**Example** `HOURON`

**See also** HOUROFF, HOURSMASH

# HOURPERCENT

**Syntax** `HOURPERCENT(`*`current, maximum`*`)`

**Use** Shows the percentage of *current* / *maximum* on the hourglass (if one is being displayed). If *current* = *maximum* = 0 then the percentage is turned off.

**Example** `HOURPERCENT(PTR#file%,length%)`

**See also** HOURON

# HOURSMASH

**Syntax** `HOURSMASH`

**Use** Sets the number of current hourglasses to zero, and resets the pointer shape to normal.

**Example** `HOURSMASH`

**See also** HOUROFF, HOURON

# ICONBAR

**Syntax** `ICONBAR(`*`sprite`*`, `*`text`*`, `*`side`*`, `*`priority`*`)`

**Use** Creates a new icon on the icon bar and returns its handle. Example *priority*‘s can be found above. *side* can mean various things, the main being -1 for right of the bar and -2 for the left. *sprite* and *text* are strings containing the sprite name and the under icon text (if wanted).

**Example**
```
newbar%=ICONBAR(”!myapp”,“”,-1,0)
```

**See also** -

# ICONHIDDEN

**Syntax** `ICONHIDDEN(`*`window`*`, `*`icon`*`)`

**Use** Returns TRUE (-1) if the icon is hidden, else returns FALSE (0).

**Example**
```
HIDEICON(mainWin,1,NOT ICONHIDDEN(mainWin,1))
```

**See also** HIDEICON

# ICONINFO

**Syntax** `ICONINFO(`*`window`*`, `*`icon`*`, `*`X`*`, `*`Y`*`, `*`width`*`, `*`height`*`)`

**Use** Returns the coordinates of the top left, the width and the height of the icon in the variables *X*, *Y*, *width* and *height* respectively.

**Example**
```
ICONINFO(mainWin,1,left%,top%,width%,height%)
```

**See also** MOVEICON

# ICONSHADED

**Syntax** `ICONSHADED(`*`window, icon`*`)`

**Use** Returns TRUE if the icon has been shaded (and so is unselectable) using SHADEICON (or created that way), or FALSE if it has not.

**Example**
```
shaded%=ICONSHADED(mainWin,0)
```

**See also** SHADEICON

# ITEMSHADED

**Syntax** `ITEMSHADED(`*`menu, item`*`)`

**Use** Returns TRUE if the item has been shaded (and so is unselectable) in the control string (using ‘G’), otherwise FALSE is returned.

**Example**
```
shaded%=ITEMSHADED(iconMenu%,0)
```

**See also** ADDITEM, CHANGEITEM

# ITEMTICKED

**Syntax** `ITEMTICKED(`*`menu, item`*`)`

**Use** Returns TRUE if the item has been ticked in the control string (using ‘T’), otherwise FALSE is returned.

**Example**
```
ticked%=ITEMTICKED(iconMenu%,0)
```

**See also** ADDITEM, CHANGEITEM

# LCASE

**Syntax** `LCASE(`*`expr`*`)`

**Use** Returns *expr* as a lowercase string, ie. any uppercase characters are changed to the corresponding lowercase letter, others are left alone.

**Example**
```
var$=LCASE(var$)
```

**See also** TRANSLATE, UCASE

# LIMIT

**Syntax** `LIMIT(`*`window`*`)`

**Use** Limits the mouse pointer to the visible workarea of *window*. If *window* = -1 then the pointer is released back to the whole screen.

**Example**
```
LIMIT(mainWin)
```

**See also** -

# LINE

**Syntax** `LINE(`*`window, X0, Y0, X1, Y1, colour`*`)`

**Use** Draws a line in *window* from *X0*, *Y0* to *X1*, *Y1*. *colour* is the 24-bit colour of the line, these are of the form &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
LINE(mainWin,0,0,200,-200,&00FF0000)
```

**See also** CLWIN

# LINK

**Syntax** `LINK(`*`pane`*`, `*`main`*`, `*`X`*`, `*`Y`*`)`

**Use** Links the window *pane* to *main* at the offset (top-left to top-left of visible area) given in *X*, *Y*. This means that when *main* is opened, closed, moved or iconised then *pane* will open, close, move or close as well. If *X* = -1 then the X-offset is taken from the positions in the "*Templates*" file, and similarly for *Y*.

**Example**
```
LINK(toolbox, mainWin,-194,0)
```

**See also** UNLINK

# LOADACK

**Syntax**
```
LOADACK
```

**Use** If a file has been double clicked or dragged to your application, this command should be called as soon as possible after you have decided that you do not want other applications to be informed. Once POLL has been called then it is too late, so if you need to use a message box use STDBOX.

**Example**
```
IF type%<>&AFF THEN ENDPROC ELSE LOADACK
```

**See also** -

# LOADTEMPLATE

**Syntax** `LOADTEMPLATE(`*`file, name`*`)`

**Use** Returns the window handle of the new window created when *name* is loaded from the Templates file *file*. Any icons which have been named can be accessed using *name_icon*.

**Example**
```
newWin%=LOADTEMPLATE(mydir$ +
".Templates","load")
```

**See also** -

# LTRIM

**Syntax** `LTRIM(`*`expr`*`)`

**Use** Returns *expr* with any leading whitespace (spaces or tabs) removed. Trailing whitespace can be removed using RTRIM and both can be removed using TRIM.

**Example**
```
var$=LTRIM(var$)
```

**See also** RTRIM, TRIM

# MAKESLIDER

**Syntax** `MAKESLIDER(`*`window, background`*`)`

**Use** Informs *WimpWorks* that the icons *background* and *background* + 1 form a slider, where *background* is the background which also determines the maximum and minimum values of the slider. *background* + 1 is the coloured part which is actually dragged. The system will then take care of updating the slider automatically and whilst the bar is being dragged your task will receive *Slider changed* events (the value passed in this event is that of *background*, **not** *background* + 1).

**Example**
```
MAKESLIDER(mainWin,mainWin_slider)
```

**See also** READSLIDER, SETSLIDER

# MEMCOPY

**Syntax** `MEMCOPY(`*`source, destination, length`*`)`

**Use** Copies *length* bytes from *source* to *destination*. The memory areas may overlap and the system will take care of this.

**Example**
```
MEMCOPY(addr%,mem%,blksize%)
```

**See also** CLAIM, RELEASE

# MESSAGE

**Syntax** `MESSAGE(`*`text`*`, `*`title`*`, `*`sprite`*`, `*`default`*`, `*`B`*`, `*`C`*`, `*`D`*`)`

**Use** Opens a multitasking message box with *text* as the message and *title* as the title. It returns an integer representing the button pressed. The lower of the two sprites is *sprite* (if this cannot be found then "error" will be used). *default*, *B*, *C* and *D* contain the text to use in the 4 buttons available (returning 1-4 respectively). If one of these strings is blank then the corresponding icon will not be shown. *default* can also be selected by pressing **Return** or **F5**, *B* by **Escape** or **F4**, *C* by **F3** and *D* can be selected by pressing **F2**. This command requires the *error* window to be present in the "*Templates*" file.

**Example**
```
choice% = MESSAGE("Erk", "Error", "warning",
"OK", "Quit", "", "")
```

**See also** CLOSEMSGBOX, STDBOX, SURE, WARNING

# MOVEICON

**Syntax** `MOVEICON(`*`window`*`, `*`icon`*`, `*`X`*`, `*`Y`*`)`

**Use** Moves *icon* to the new position (top-left) in *window*. Difficulties may be encountered if icons have been hidden or shaded (and an older version of the WIMP is being used).

**Example**
```
MOVEICON(mainWin,0,newX%,newY%)
```

**See also** COPYICON, DELETEICON

# MOVEWINDOW

**Syntax** `MOVEWINDOW(`*`window, X, Y`*`)`

**Use** Moves the top-left corner of the visible work area of *window* to *X*, *Y* (OS coordinates).

**Example**
```
MOVEWINDOW(mainWin,640,512)
```

**See also** CENTREWIN

# NEWMENU

**Syntax** `NEWMENU(`*`expr`*`)`

**Use** Creates a new (empty) menu structure with the string *expr* as the title and returns a pointer to the menu block, this value is the menu handle.

**Example**
```
newMenu%=NEWMENU(”The Title”)
```

**See also** ADDITEM, DELETEMENU

# OPENMENU

**Syntax** `OPENMENU(`*`menu, X, Y`*`)`

**Use** Opens *menu* at the OS coordinates *X*, *Y*. If *X* = -1 then the current mouse pointer position is used instead, and similarly for *Y*. If *menu* = -1 then any open menu is closed.

**Example**
```
OPENMENU(winMenu%,-1,-1)
```

**See also** OPENPOPUP

# OPENPOPUP

**Syntax** `OPENPOPUP(`*`menu, window, icon`*`)`

**Use** Opens *menu* from the popup icon specified in *window* and *icon*, the menu's top-left corner is at the top-right corner of the icon.

**Example**
```
OPENPOPUP(winMenu%,mainWin,mainWin_popup)
```

**See also** OPENMENU

# OPENQUERY

**Syntax** `OPENQUERY(`*`window`*`)`

**Use** Returns TRUE if *window* is open, or FALSE if it is not.

**Example**
```
open%=OPENQUERY(mainWin)
```

**See also** CLOSEWINDOW, OPENWINDOW

# OPENSAVE

**Syntax** `OPENSAVE`

**Use** Opens the ‘save as’ box at the current mouse pointer position, or moves it there if it is already open.

**Example**
```
OPENSAVE
```

**See also** CLOSESAVE

# OPENWINDOW

**Syntax** `OPENWINDOW(`*`window`*`)`

**Use** Opens *window* in its last open position (or the position in the "*Templates*" file if not previously opened) if the *Window opening* event is undefined or returns true.

**Example**
```
OPENWINDOW(mainWin)
```

**See also** CENTREWIN, CLOSEWINDOW, OPENQUERY

# OSVAR

**Syntax** `OSVAR(`*`expr`*`)`

**Use** Returns the contents of the string system variable held in *expr*. The empty string ("") is returned if the variable does not exist.

**Example**
```
wimpscrap$=OSVAR("Wimp$Scrap")
```

**See also** -

# POLL

## Syntax

```
POLL
```

## Use

Returns the action code returned from the application's Wimp_Poll code after all the system events have been taken care of. This can be used inside a loop to ensure that your application multitasks. The WIMP control block can be found using TASKINFO.

The full list of action codes are:

| | |
|---|---|
| 0 | Null |
| 1 | Redraw window |
| 2 | Open window |
| 3 | Close window |
| 4 | Pointer leaving window |
| 5 | Pointer entering window |
| 6 | Mouse click |
| 7 | User drag box |
| 8 | Key pressed |
| 9 | Menu selection |
| 10 | Scroll request |
| 11 | Lose caret |
| 12 | Gain caret |
| 13 | Pollword non-zero |
| 14-16 | Reserved |
| 17 | User message |
| 18 | User message recorded |
| 19 | User message acknowledge |

## Example

```
BUSYON:REPEAT UNTIL POLL=0:BUSYOFF
```

## See also

TASKINFO

# READICON

**Syntax** `READICON(`*`window, icon`*`)`

**Use** Returns the string contents of the specified icon.

**Example**
```
var$=READICON(mainWin,0)
```

**See also** SETICON

# READSLIDER

**Syntax** `READSLIDER(`*`window, background`*`)`

**Use** Returns the current value of the slider whose background icon is that in *background*. The possible values are in the range 0-100, representing the percentage of the slider to the right. The value can be set using SETSLIDER or by the user physically dragging the bar.

**Example**
```
volume%=READSLIDER(mainWin,mainWin_slider)
```

**See also** MAKESLIDER, SETSLIDER

# READSTATE

**Syntax** `READSTATE(`*`window`*`, `*`icon`*`)`

**Use** Returns TRUE if the icon has been selected by the user or using SETSTATE, otherwise it returns FALSE.

**Example**
```
selected%=READSTATE(mainWin,0)
```

**See also** SETSTATE

# READTITLE

**Syntax** `READTITLE(`*`window`*`)`

**Use** Returns a string containing the current title of *window*.

**Example**
```
title$=READTITLE(mainWin)
```

**See also** SETTITLE

# RECTANGLE

**Syntax** `RECTANGLE(`*`window`*`, `*`X`*`, `*`Y`*`, `*`width`*`, `*`height`*`, `*`colour`*`)`

**Use** Draws a rectangle outline in *window* with the top-left corner at *X*, *Y* and width and height as in *width* and *height* respectively. *colour* represents the 24-bit colour in the form &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
RECTANGLE(mainWin,0,0,200,300,&00FF8800)
```

**See also** CLWIN, RECTANGLEFILL

# RECTANGLEFILL

**Syntax** `RECTANGLEFILL(`*`window`*`, `*`X`*`, `*`Y`*`, `*`width`*`, `*`height`*`, `*`colour`*`)`

**Use** Draws a filled rectangle in *window* with the top-left corner at *X*, *Y* and width and height as in *width* and *height* respectively. *colour* represents the 24-bit colour in the form &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
RECTANGLEFILL(mainWin,0,0,200,300,&00FF8800)
```

**See also** CLWIN, RECTANGLE

# REDEFINE

**Syntax** `REDEFINE(old, new, type)`

**Use** Renames the name of the function (*type* = &AF) or procedure (*type* = &F2) from *old* to *new* allowing you to supply a new subroutine of name *old* which may call the original by calling *new*. If the length of *new* < *old* then *new* is padded with underscores, it is also truncated to the length of *old* if it is too long. Only recommended for advanced users.

**Example**
```
REDEFINE(”openwindow”,“beepwindow”,&F2)
```

**See also** -

# REDRAW

**Syntax** `REDRAW(window)`

**Use** Redraws the contents of *window* and ensures that it is upto date. **Not** normally needed as *WimpWorks* takes care of all graphics handling but may occasionally be useful.

**Example**
```
REDRAW(mainWin)
```

**See also** CLWIN

# RELEASE

**Syntax** `RELEASE(`*`address`*`)`

**Use** Frees the memory pointed to by *address* and returns it back to the system, if *address* was originally claimed using CLAIM. *address* must also be a variable and will be reset to -1.

**Example**
```
RELEASE(pointer%)
```

**See also** CLAIM

# REPLY

**Syntax** `REPLY(`*`task, reference, reply`*`)`

**Use** Sends an ActiveApps reply to the task with name *task*. The *reference* number was returned to *task* by COMMAND. *reply* is a string containing the reply.

**Example**
```
REPLY(caller$,ref%,"Pong")
```

**See also** COMMAND

# RESIZEICON

**Syntax** `RESIZEICON(`*`window`*`, `*`icon`*`, `*`dX`*`, `*`dY`*`)`

**Use** Resizes the icon by the changes passed in *dX* and *dY* (OS units). If *dX* > 0 then the right hand edge of the icon will move right, otherwise the same edge will move left, similarly for *dY* (bottom edge).

**Example**
```
RESIZEICON(mainWin,0,32,32)
```

**See also** ICONINFO

# ROUND

**Syntax** `ROUND(`*`expr`*`, `*`places`*`)`

**Use** Returns *expr* rounded off to *places* decimal places.

**Example**
```
IF ROUND(3.20873,3) = 3.209 THEN ...
```

**See also** CEILING, INT

# RTRIM

**Syntax** `RTRIM(`*`expr`*`)`

**Use** Returns *expr* with any trailing whitespace (spaces or tabs) removed. Leading whitespace can be removed using LTRIM and both can be removed using TRIM.

**Example**
```
var$=RTRIM(var$)
```

**See also** LTRIM, TRIM

# SETCARET

**Syntax** `SETCARET(`*`window, icon, position`*`)`

**Use** Places the caret (the I-shaped input cursor) in the specified icon at index *position* within the icon. If *window* = -1 then the caret is hidden; *icon* = -1 means that the input focus goes to the background of the window and if *position* = -1 then the caret goes to the end of the string.

**Example**
```
SETCARET(mainWin,3,-1)
```

**See also** READCARET

# SETICON

**Syntax** `SETICON(window, icon, expr)`

**Use** Sets the text (or sprite if a sprite only icon) to the string *expr* in the specified icon.

**Example** `SETICON(mainWin,0,“Hello world”)`

**See also** CLEARALL, READICON

# SETICONCOLOUR

**Syntax** `SETICONCOLOUR(`*`window`*`, `*`icon`*`, `*`foreground`*`, `*`background`*`)`

**Use** Sets the colours of the specified icon to those passed in *foreground* and *background* - the colours are in the range 0-15 and represent the standard WIMP palette. If *foreground* = -1 then the foreground colour is not changed, and similarly for *background*.

**Example** `SETICONCOLOUR(mainWin,0,1,7)`

**See also** SETVALID

# SETSAVE

**Syntax** `SETSAVE(`*`name`*`, `*`sprite`*`)`

**Use** Sets the default filename (*name*) and sprite to show and drag (*sprite*) in the ‘save as’ box to use when it is next opened using OPENSAVE or from a menu.

**Example**
```
SETSAVE(”TextFile”,“file_fff”)
```

**See also** CLOSESAVE, OPENSAVE, SETICON

# SETSIZE

**Syntax** `SETSIZE(`*`window`*`, `*`width`*`, `*`height`*`)`

**Use** Changes the total size of *window* to *width* and *height* (OS units). If the window is open then the **visible** size is also changed, as well as the total size. If *width* = -1 then the width is not changed, and similarly for *height*.

**Example**
```
SETSIZE(mainWin,384,-1)
```

**See also** WINDOWSIZE

# SETSLIDER

**Syntax** `SETSLIDER(`*`window`*`, `*`background`*`, `*`expr`*`)`

**Use** Sets the value of the slider (whose background icon = *background*) to *expr*, if it is in the range 0-100, representing the percentage of the slider to the right. The value can be read using READSLIDER and is passed in the *Slider changed* event.

**Example**
```
SETSLIDER(mainWin,mainWin_slider,50)
```

**See also** MAKESLIDER, READSLIDER

# SETSTATE

**Syntax** `SETSTATE(`*`window`*`, `*`icon`*`, `*`expr`*`)`

**Use** If *expr* is true (ie. non-zero) then the icon is selected, otherwise it is deselected. The current state can be read using READSTATE and can be seen by the user on screen (usually).

**Example**
```
SETSTATE(mainWin,2,TRUE)
```

**See also** READSTATE

# SETTITLE

**Syntax** `SETTITLE(`*`window, expr`*`)`

**Use** Sets the title of *window* to the string *expr*. If the length of *expr* is longer than the maximum length then it is cropped to fit. **NB**. *window* must have an indirected title bar otherwise this will fail.

**Example**
```
SETTITLE(mainWin,filename$)
```

**See also** READTITLE

# SETVALID

**Syntax** `SETVALID(`*`window, icon, expr`*`)`

**Use** Sets the validation string of *icon* to *expr*. If the length of *expr* > current length then *expr* is truncated to fit.

**Example**
```
SETVALID(mainWin,2,“sfile_fff”)
```

**See also** SETICON

# SHADEICON

**Syntax** `SHADEICON(`*`window`*`, `*`icon`*`, `*`expr`*`)`

**Use** If *expr* is true (ie. non-zero) then the specified icon is shaded (and so is made unselectable), otherwise it is unshaded.

**Example**
```
SHADEICON(mainWin,0,TRUE)
```

**See also** ICONSHADED

# SHELL

**Syntax** `SHELL(`*`expr`*`)`

**Use** Starts a new WIMP task by executing the command *expr*. This call returns when the new task completes or calls Wimp_Poll - whichever is sooner.

**Example**
```
SHELL("Run Resources:$.Apps.!Help")
```

**See also** -

# SPRITE

**Syntax** `SPRITE(`*`window`*`, `*`X`*`, `*`Y`*`, `*`ratio`*`, `*`scale`*`, `*`address`*`, `*`name`*`)`

**Use** Draws the sprite called *name* (from the sprite area pointed to by *address*) in *window* with the top-left at *X*, *Y*. *ratio* is the horizontal size : vertical size ratio and *scale* is the scaling of the sprite - both are in the range 0-100.

**Example**
```
SPRITE(mainWin,0,0,100,50,sprite%,“logo”)
```

**See also** CLWIN

# STARTDRAG

## Syntax

`STARTDRAG(`*`type`*`,` *`subroutine`*`,` *`window`*`,` *`X0`*`,` *`Y0`*`,` *`width0`*`,` *`height0`*`,` *`X1`*`,` *`Y1`*`,` *`width1`*`,` *`height1`*`,` *`expr`*`)`

## Use

This call starts a new drag operation, *type* indicates the type of drag:

1 Fixed box
2 Rubber box
3 Reserved
4 Sprite

*subroutine* is what to call when the drag is complete and its parameters **must** be of the form `(x0,y0,x1,y1)`. *window* is the window on which to base the coordinates, -1 if the coordinates are absolute.

*X0*, *Y0*, *width0* and *height0* define the size and position of the start position of the box to drag. Similarly *X1*, *Y1*, *width1* and *height1* define the size and position of the bounding box (if they are all zero then it is taken to be the screen).

*expr* is a string containing any extra parameters, this is currently only used by type 4, where *expr* contains the sprite name to drag.

## Example

```
STARTDRAG(1, “PROCdrag_stop”, mainWin, x0%,
y0%, width%, height%, 0, 0, 0, 0, “”)
```

## See also

ENDDRAG

# STDBOX

## Syntax

`STDBOX(`*`text`*`, `*`title`*`, `*`flags`*`)`

## Use

Opens a standard system message box (which is **not** multitasking) and returns the button pressed (1 = **OK**, 2 = **Cancel**). *flags* consists of:

| ***Bit*** | ***Meaning when set*** |
|---|---|
| 0 | **OK** button |
| 1 | **Cancel** button |
| 2 | Highlight **Cancel** (or **OK** if no **Cancel**) |
| 4 | Do not prefix *title* with “Error from “ |
| 5 | Return immediately with 0 and box open |
| 6 | Simulate click in box according to bits 0-1 |
| 7 | Do not beep |
| 8 | Use categories (not supported) |
| 9-11 | Category (advanced users only) |

## Example

```
choice%=STDBOX(”Error”,“WimpWorks”,%101)
```

## See also

MESSAGE, SURE, WARNING

# SUBST

**Syntax** `SUBST(`*`token`*`, `*`%0`*`, `*`%1`*`, `*`%2`*`, `*`%3`*`)`

**Use** Returns *token* with the %0-%3 in the token replaced with the relevant string. The token is looked for in the "*Messages*" file and an empty string is returned if it could not be found.

**Example**
```
message$=SUBST(”Error”, TOKEN(”Memory”), “”, “”, “”)
```

**See also** TOKEN

# SURE

**Syntax** `SURE(`*`text`*`, `*`title`*`)`

**Use** Opens a multitasking error box with two buttons - **OK** and **Cancel** and returns TRUE if **OK** is selected (the default button) or FALSE if **Cancel** is. The "question" sprite is used if present. If the "error" window is not present, a standard error box is used.

**Example**
```
IF SURE(”Are you sure you want to continue?”, “Query from WimpWorks”) THEN ...
```

**See also** CLOSEMSGBOX, MESSAGE, STDBOX, WARNING

# SYSDATE

**Syntax** `SYSDATE`

**Use** Returns the system date in the form ***dd mon yyyy***.

**Example** `SETICON(mainWin,0,SYSDATE)`

**See also** SYSTIME

# SYSTIME

**Syntax** `SYSTIME`

**Use** Returns the system time in the form ***hh*:*mm*:*ss***.

**Example** `SETICON(mainWin,1,SYSTIME)`

**See also** SYSDATE

# TASKHANDLE

**Syntax** `TASKHANDLE(`*`expr`*`)`

**Use** Returns the unique numeric task handle of the task. *expr* must be identical to the string name in the *Task Manager*'s task list. If there is more than one task with *expr* as its name, the handle of the first will be returned. 0 is returned if no task with that name could be found.

**Example**

```
handle%=TASKHANDLE(”WimpWorks”)
```

**See also** TASKNAME

# TASKNAME

**Syntax** `TASKNAME(`*`expr`*`)`

**Use** Returns a string containing the name (as it appear in the *Task Manager*'s list) of the task, whose handle = *expr*.

**Example**

```
IF TASKNAME(val%)=”WimpWorks” THEN ...
```

**See also** TASKHANDLE

# TASKINFO

## Syntax

`TASKINFO(`*`expr`*`)`

## Use

Returns an integer whose value depends upon *expr*:

| | |
|---|---|
| &00 | Wimp version * 100 |
| &01 | Pointer to string containing application's directory |
| &02 | Application's sprite area |
| &03 | Iconbar icon handle |
| &04 | Task handle |
| &05 | Current WimpSlot value (in kbytes) |
| &06 | Pointer to 256 byte WIMP block |
| &07 | Pointer to poll word |
| &08 | Pointer to start of heap |
| &09 | Size of heap (bytes) |
| &0A<br>&99 | Reserved for future expansion (inclusive) |
| &A0 | 'Info box' window handle |
| &A1 | 'Save as' window handle |
| &A2 | 'Error' window handle |

## Example

```
taskdir$=ZSTRING(TASKINFO(&01))
```

## See also

-

# TEXT

**Syntax** `TEXT(`*`window`*`, `*`X`*`, `*`Y`*`, `*`text`*`, `*`colour`*`)`

**Use** Draws *text* in the system font (top-left corner at *X*, *Y*) in *window* with the 24-bit colour *colour. colour* is in the form &BBGGRR00, eg. &FF00FF00 is magenta.

**Example**
```
TEXT(mainWin,0,0,“Hello world”,&0000FF00)
```

**See also** CLWIN

# TOKEN

**Syntax** `TOKEN(`*`expr`*`)`

**Use** Returns the token *expr* from the “*Messages*” file without any substitution. The empty string (””) is returned if the token could not be found.

**Example**
```
message$=TOKEN(”NoMemory”)
```

**See also** SUBST

# TRANSLATE

**Syntax** `TRANSLATE(`*`expr, source, destination`*`)`

**Use** Returns *expr* with any characters from *source* converted to the corresponding character in *destination*, eg. TRANSLATE(”ABCD”,“AC”,“XY”) = “XBYD”.

If there is no corresponding character in *destination* then the character is left unchanged.

**Example**
```
var$=TRANSLATE(var$,CHR$9,CHR$32)
```

**See also** -

# TRIM

**Syntax** `TRIM(`*`expr`*`)`

**Use** Returns *expr* with any leading or trailing whitespace (spaces or tabs) removed. Leading whitespace can also be removed using LTRIM and trailing using RTRIM.

**Example**
```
var$=TRIM(var$)
```

**See also** LTRIM, RTRIM

# UCASE

**Syntax** `UCASE(`*`expr`*`)`

**Use** Returns *expr* with any lowercase letters converted to uppercase and any others left alone.

**Example**
```
IF UCASE(”abCdE”)=”ABCDE” THEN ...
```

**See also** LCASE, TRANSLATE

# UNLINK

**Syntax** `UNLINK(`*`pane, main`*`)`

**Use** Detaches *pane* from *main* (if they have been linked using LINK), but does not close it if they are open.

**Example**
```
UNLINK(toolbox,mainWin)
```

**See also** LINK

# WARNING

**Syntax** `WARNING(`*`text, title`*`)`

**Use** Opens a multitasking box with just an **OK** button. Uses the "information" sprite if present.

**Example**
```
WARNING(TOKEN("E1"),"Message from WimpWorks")
```

**See also** CLOSEMSGBOX, MESSAGE, STDBOX, SURE

# WGET

**Syntax** `WGET(`*`file`*`)`

**Use** Returns the next 4-byte word from *file* and increments the current pointer (readable using PTR#*file*). Similar to BASIC's BGET.

**Example**
```
word%=WGET(file%)
```

**See also** BGET, WPUT

# WINDOWPOS

**Syntax** `WINDOWPOS(`*`window`*`, `*`X`*`, `*`Y`*`, `*`scrollX`*`, `*`scrollY`*`)`

**Use** Fills the variables *X*, *Y*, *scrollX* and *scrollY* with the current X and Y positions on screen and the current values of the horizontal and vertical scroll bars respectively, all in OS units.

**Example**
```
WINDOWPOS(mainWin,x%,y%,scrollx%,scrolly%)
```

**See also** WINDOWSIZE

# WINDOWSIZE

**Syntax** `WINDOWSIZE(`*`window`*`, `*`width`*`, `*`height`*`)`

**Use** *width* and *height* are filled with the current size of *window* on screen (OS units).

**Example**
```
WINDOWSIZE(mainWin,width%,height%)
```

**See also** WINDOWPOS

# WPUT

**Syntax** `WPUT(`*`file, word`*`)`

**Use** Puts the 4-byte word *word* into *file* at the current position (writable using PTR#*file*) - this is similar to BASIC's BPUT.

**Example**
```
WPUT(file%,&4B535462)
```

**See also** BPUT, WGET

# ZSTRING

**Syntax** `ZSTRING(`*`address`*`)`

**Use** Returns the string terminated by a control character (< 32) starting at the address *address*.

**Example**
```
var$=ZSTRING(mem%+28)
```

**See also** -

# Hints & Tips

## Standalone window editor

The *Window Editor* can be used as a separate product to edit other applications' template files. To make a copy of it open the directory !WimpWorks.Resources.Editors.!AAAB_Wind and there is an application called *!WindowEd*. Simply copy this to another location and run it. *WindowEd* is a **freeware** product and may be distributed separately to *WimpWorks* - however the !WimpWorks application **cannot** be distributed. *WindowEd* contains some code by Dick Alstein.

## Converting WimpWorks apps to WWv2

Old applications produced with *WimpWorks 1* cannot directly be edited by *WimpWorks 2* - many commands remain the same, however there have been many changes to the file format making an automated convertor difficult. However if you have any difficulties converting an old application, please send it to Jaffa Software and we will do our best to convert it - free of charge.

However, to make it easier - your old application's windows should be **copied** to the *Window Editor* as copying the old "*Templates*" file over the new one will mean that the extended error box, the save box and the program information box will have been lost.

## Deleting the iconbar icon

This is simply a matter of adding a command to your *Starting* event:

```
DELETEICON(-2, TASKINFO(wwIconbarHandle))
```

or, if the *Starter Pack WEM* is **not** being included:

```
DELETEICON(-2, TASKINFO(3))
```

# Writable menu items

To change, for example, the bottom entry in *mymenu%* - which has a total of 5 items to allow strings of upto 63 characters to be entered, this code would need to be put in the *Starting* event:

```
writable%=CLAIM(64) : CHANGEITEM(mymenu%,4,“The default text”,“W(writable%,64)”)
```

The value of the writable item can be accessed by *$writable%* and can be written to in the same way. *writable%* may, of course, only be used for one item, unless you want the same text to appear in two separate items, in which case it should **not** be re-CLAIMed.

# ActiveApps aliases

To accept ActiveApps messages to more than one task name you need to decode the ActiveApps messages (&50300 and &50301) in the *POLL called* event - for an example of this see the *AAterm* example.

# Distributing your software

All software produced with *WimpWorks* can be distributed by you as you see fit without any royalty payments to Jaffa Software. However, no attempt, or encouragement, may be made to reverse engineer or decompile any application’s “*!RunImage*”.

Your freeware application’s can be sent to Jaffa Software by mail or email for inclusion on our webpages or a compilation disk, if you wish to.

*!WindowEd* is freeware and may be distributed separately from *WimpWorks*. However, no other part or component of the *!WimpWorks* directory may be distributed, without express permission from Jaffa Software.

# Jaffa Software

Our email address and webpages are at:

`info@jaffasoft.co.uk`
`http://www.jaffasoft.co.uk/`

There is a mailing list for queries and discussions about WimpWorks and details on this can be found at our webpages above.

Our postal address is:

28 Grange Farm Drive
Stockton
Southam
CV47 8FT
United Kingdom

**Phone:** +44 (0)7010 704228

# Index

## T



## U



## W



## Z