

TWimp Reference Guide

Purpose of TWimp

The TWimp module helps you create Risc OS Wimp applications using text descriptions. It is perfect for people who quickly want to make a mock example of their program, for beginners to making multitasking applications and those who want to concentrate on their own code rather than the complicated details of the Risc OS Wimp.

As long as your programming language supports SWI calls then TWimp can be used. Calls remain the same between programming languages so moving from one language such as BASIC to something different like C++ or Python does not require that you learn a new library.

The T stands for text and the Wimp stands for the usual Risc OS acronym of windows, icons, menus and pointers.

Windows, menus and icons can have names that you choose and you can then use the assigned name in your program. These will be called 'items'.

Items can be created by loading in a text file where you can bulk create your items separately from your program.

An example SWI to update an item to change the sprite and text at the same time would be:

```
SWI "TWimp_Update", "ItemName",  
    "sprite: 'BlueSprite' text: 'Hello World'"
```

The above would update ItemName. The item name could refer to an iconbar icon, a window icon or something else. The update part of the text has a large number of changes you can make. Colours, alignments, text, sprite, borders, positions, greyed and selected statuses and much more.

Many simple tasks can be performed without any programming at all, except for the basic TWimp loop example as described below.

Most features will work from Risc OS 3 or later.

Contents

Table of Contents

TWimp Reference Guide.....	1
Purpose of TWimp.....	1
Contents.....	2
TWimp Poll Loop Example: BASIC.....	5
TWimp Poll Loop Example: Python.....	6
TWimp Poll Loop Example: Lua.....	7
TWimp Poll Loop SWIs.....	8
TWimp_Start.....	8
TWimp_StartOnce.....	8
TWimp_End.....	8
TWimp_PollSetup.....	8
TWimp_PollAction.....	8
Loading an Item File.....	9
TWimp_Load.....	9
Basic Item File Structure.....	10
TWimp Items.....	12
TWimp Keywords.....	13
TWimp Action Keywords.....	18
Example Windows.....	21
Example Icons.....	22
Example Menus.....	25
Opening and Closing Items.....	26
TWimp_Open.....	26
TWimp_OpenAt.....	26
TWimp_Close.....	26
Updating Items from your Program.....	27
TWimp_Update.....	27
TWimp_UpdateText.....	27
TWimp_UpdateSprite.....	27
TWimp_UpdateValue.....	27
TWimp_UpdateNumber.....	27
Reading Item Data from your Program.....	28
TWimp_Read.....	28
TWimp_ReadText.....	28
TWimp_ReadSprite.....	28
TWimp_ReadValue.....	28
Reading Data from the Wimp.....	29
TWimp_PollData.....	29
Groups and Frames.....	33
Timer Events.....	35
TWimp_AddTimer.....	35
TWimp_RemoveTimer.....	36
Saving and Loading Files.....	37
Loading Files.....	37

TWimp_Loaded.....	38
Loading Files from Filer Double-Clicks.....	38
Loading a File When Your Application is Not Loaded.....	38
Saving Files.....	39
TWimp_Saved.....	40
Save Window Options and Radio Buttons.....	40
Save Window Style.....	40
Setting Windows to be File Windows.....	40
Opening and Closing Files.....	41
TWimp_MiscOp, “Open-w” and “Open-r”.....	42
TWimp_MiscOp, “Close”.....	42
Saving and Loading Options.....	43
TWimp_SaveOptions.....	43
TWimp_LoadOptions.....	43
Dragging Things (Fixed Sized Box).....	44
Events.....	46
TWimp_Event.....	46
Events with Parameters.....	47
Dynamically Creating and Deleting TWimp Items.....	48
TWimp_Create.....	48
Variables.....	50
SWI TWimp_MiscOp, “add-var”.....	50
SWI TWimp_MiscOp, “add-var-perm”.....	50
SWI TWimp_MiscOp, “del-vars”.....	50
SWI TWimp_MiscOp, “del-vars-perm”.....	51
Using Variables in Your Program.....	51
Using Variables in Actions.....	51
Reading Variables Back into Your Program.....	52
SWI TWimp_MiscOp, “read-var”.....	52
SWI TWimp_MiscOp, “read-var-value”.....	52
Global Settings.....	53
Language Support.....	54
Swapping the Language.....	54
Using Tags.....	54
Scripts.....	56
TWimp_Action (Run Action).....	56
TWimp_Action (Simulation).....	56
Copying Designs.....	57
Icon Positioning to a Window.....	58
Position an Icon to a Window Corner.....	58
Automatically Updating if the Window Size Changes.....	58
Keep an Icon in the Visible Area.....	58
Advanced Icon Positioning.....	59
Last Icon Positioning.....	59
Icon Relative Positioning to Specific Icon.....	59
Frame Icon Positioning.....	60
Group Icon Positioning.....	61
Pre and Post Icon Positioning.....	61
Bounding Icon Positioning.....	61
Error Messages.....	62

TWimp_Error.....	62
Claiming Memory.....	64
SWI TWimp_MiscOp, "Claim".....	64
SWI TWimp_MiscOp, "Release".....	64
Programming in C, Using FPE and Poll Words.....	65
Programming in C.....	65
TWimp_ReturnZero.....	65
Using the FPE.....	65
TWimp_MiscOp, "FPE".....	65
Poll Words.....	65
TWimp_MiscOp, "Poll-Word".....	65
Debugging and Help.....	66
TWimp_MiscOp, "Debug-names".....	66
Sixteen Colour Modes.....	66
General Information.....	67
Reading the Screen Size.....	67
TWimp_MiscOp, "Screen-size".....	67
My Task Handle.....	67
TWimp_MiscOp, "Task-handle".....	67
Renaming Items.....	67
TWimp_MiscOp, "Rename".....	67
Swapping the Names of Items.....	67
TWimp_MiscOp, "Swap-name".....	67
Working with Risc OS 2 and Risc OS 3.11.....	67
TWimp_MiscOp, "Poll-SWI".....	68
TWimp_MiscOp, "Set-task".....	68
SWI Numbers.....	70

TWimp Poll Loop Example: BASIC

A basic program would consist of the following with this example in BASIC. The SWI calls in BASIC are done using the SYS command.

```
SYS "TWimp_Start", "My Application"
quit = 0
REPEAT

    REM TWimp Poll Loop
    SYS "TWimp_PollSetup" TO timer%,b%
    SYS "Wimp_PollIdle",timer%,b% TO reason%,b%,sender%
    SYS "TWimp_PollAction",reason%,b%,sender% TO reason$

    REM Respond to reason$
    CASE reason$ OF
        WHEN "QUIT" : quit = 1
    ENDCASE

UNTIL quit = 1

SYS "TWimp_End"
END
```

The above program will create an application if the TWimp module is loaded. You will only see My Application listed in the task manager. Nothing else will be shown. Selecting quit from the task manager will send QUIT to your program in reason\$ and the program responds accordingly by setting quit to one, which ends the loop and then calls TWimp_End.

TWimp Poll Loop Example: Python

A simple TWimp starter program in Python.

```
import swi

swi.swi("TWimp_Start","s","Python TWimp Example")

quit = 0
while quit == 0:

    // Standard poll loop
    a,b,c=swi.swi("TWimp_PollSetup",";III")
    rcode,bk,sd=swi.swi("Wimp_PollIdle","III;III",a,b,c)
    reason=swi.swi("TWimp_PollAction","III;s",rcode,bk,sd)

    // Handle reason string here
    # reason contains the event. It is a string.
    if reason == "QUIT":
        quit = 1

swi.swi("TWimp_End",";")
```

TWimp Poll Loop Example: Lua

```
#!/ lua
local sys,$ in riscos

sys("TWimp_Start","Lua TWimp Example")

repeat
  a,b,c = sys("TWimp_PollSetup")
  rcode,bk,sd = sys("Wimp_PollIdle",a,b,c)
  reason = sys("TWimp_PollAction",rcode,bk,sd)
  reason$ = ${reason}

until reason$ == "QUIT"

sys("TWimp_End")
```

TWimp Poll Loop SWIs

You will need the following SWI calls to run a basic TWimp application.

TWimp_Start

TWimp_StartOnce

Entry: R0 Application name

Starts a TWimp application. You will require a TWimp loop to respond to events and to end with TWimp_End to form a complete application.

TWimp_StartOnce is identical to TWimp_Start but will cause an error if an application is already running with the same name.

TWimp_End

This ends your application. You should not call TWimp SWI calls after you have called TWimp_End.

TWimp_PollSetup

Entry: R0 Should be zero (reserved for future use)
Exit: R0 Parameter R0 for Wimp_PollIdle
 R1 Parameter R1 for Wimp_PollIdle
 R2 Parameter R2 for Wimp_PollIdle

Part of the TWimp poll loop as described in the Poll Loop chapter. It returns parameters that you should pass to the real Wimp call Wimp_PollIdle. For most programs, Wimp_PollIdle is the only real Wimp call you will require and you will use TWimp calls for everything else.

TWimp_PollAction

Entry: R0 Parameter from R0 that Wimp_PollIdle gave you
 R1 Parameter from R1 that Wimp_PollIdle gave you
 R2 Parameter from R2 that Wimp_PollIdle gave you
Exit: R0 Reason text pointer that you should respond to
 R1 Reason code number from Wimp

Part of the TWimp poll loop as described in the Poll Loop chapter. It performs actions for you so your program does not have to deal with intricate parts of the Wimp. It should be called after Wimp_PollIdle. Your loop should be TWimp_PollSetup, Wimp_PollIdle and then TWimp_PollAction.

Once you have set the poll loop up, the reason text pointer will give your program a string in R0 that your program should respond to accordingly.

Loading an Item File

An item file can contain definitions for many TWimp items, including windows, icons, menus, iconbar icons and can be loaded in with one simple SWI.

We need to create a TWimp file so we can load it in. This is just a text file that you can create in any text editor]. A very simple example input file could be:

```
<iconbar>
    sprite : "!edit"
    on-click : -open mainwindow
    on-menu  : -open mainmenu

<window mainwindow>
    title : "Main Window"
    width : 400 height : 200

<icon>
    text : "Hello World"
    width : 400

<menu mainmenu>
    title : "Main Menu"
<menuitem>
    text : "Quit"
    on-click : QUIT
```

The blank spaces can be spaces or tabs, it does not matter. TWimp is very flexible.

Save this as InputFile. These are loaded in with the TWimp_Load SWI.

TWimp_Load

Entry: R0 Filename of text file to load in containing TWimp definitions
 R1 Filename of spritefile to use. Zero or null filename if no sprites.

Loads in the filename and the optional spritefile and creates all the items for you.

Add this SWI to your program after the TWimp_Start line, pointing to the correct file you created and your example program can now load in this new file we have created. If you are using C++ or Python, ensure that R1 is zero on entry.

```
REM Change filename to point to your file
SYS "TWimp_Load", "SDFS:$.InputFile"
```

Basic Item File Structure

Each item in the filename is started with the angled brackets < and > and inside these brackets is the TWimp defined type. Basic defined types include 'iconbar', 'window', 'icon', 'menu' and 'menuitem' as shown in the example. There are many more available.

```
// Create an empty iconbar icon
<iconbar>
```

The defined types are optionally followed by the item name afterward. This is a name you pick and you can tell TWimp to do things with that name. We have created a window and given it the name mainwindow and also the menu has the name mainmenu.

```
// Create an empty iconbar icon called ib
<iconbar ib>
```

After the angled brackets we define the item. The iconbar item has a sprite which has been 'borrowed' from !Edit. The window has a title of "Main Window".

```
// Create an iconbar icon called ib with a sprite
<iconbar ib> sprite : "!Edit"
```

The iconbar also has a couple of actions defined. Both on-click and on-menu are action definitions. If you run your program with the above definition file you will see the !Edit icon on the taskbar. Clicking on that iconbar icon opens mainwindow and clicking menu opens the mainmenu.

You may have noticed that the menuitem also has an on-click action. It is simply defined as QUIT.

The first action types have a subtract sign in front of them. This tells TWimp that it should try and handle this. The menuitem does not have the subtract sign. That text will be returned to your program. In the BASIC example this is in reason\$ returned from the TWimp_PollAction SWI and we already are responding to QUIT to exit the program so selected Quit from the menu will cause the program to exit.

```
// A subtract sign indicates TWimp should handle this
// TWimp instruction
on-click : -open itemname

// Send DoSomething back to your program
on-click : DoSomething
```

Each definition should be the TWimp keyword followed by the colon and then the value(s). Some keywords do not require a parameter and do not need the colon. It is not case-sensitive, hyphens are optional, spaces and tabs can be

before and/or after the colon (or equals) symbol and you can put many of them on the same line, like width and height is for the window.

Strings should be in quotes, either " or ' and should be terminated by the same type of quote. To include a quote of the type you started the string with, use a double quote of the same type, such as 'You know it's fun'. By allowing either type of quote TWimp makes it a little easier for a variety of programming languages.

Unrecognised keywords will be ignored and you will not receive an error.

Many spelling errors and alternative versions are available. Some keywords do the same thing. COLOUR and fg-color will do the same thing, changing the foreground colour.

TWimp Items

<iconbar>	Creates an icon on the iconbar
<window>	Creates a standard window
<transient>	Creates a window that opens menu-like (also <menuwindow>)
<icon>	Creates a standard icon in the last defined window
<label>	Creates a label icon
<data>	Creates an icon that displays data (also <display>)
<pop-up>	Creates an icon that has a pop-up menu (also <drop-down>)
<number>	Creates icon that contains a whole number. Arrows optional.
<slider>	Creates a slider icon, horizontal. Arrows and number optional.
<sliderv>	Creates a slider icon, vertical. Arrows and number optional.
<button>	Creates an icon that acts as a button
<action>	Creates an icon that acts as the action button for the window
<ok>	Creates an icon that acts as the OK button (also <default>)
<cancel>	Creates an icon that is the cancel button
<savebutton>	Creates an icon that is the save button for options
<writable>	Creates a writeable icon (also <writeable>)
<option>	Tick icon
<radio>	Radio button choice
<frame>	Creates a set of icons to make a frame around icons
<up>	Up icon (also <down>, <left> and <right>)
<pane>	Insert another window in this one (also <subwindow>)
<toolbar>	Use another window as a toolbar in this one
<menu>	Creates a menu
<menuitem>	Creates a menu choice in the last created menu (also <item>)
<about>	Creates a window that is a standard "About this program"
<save>	Creates a standard save window
<quit>	Creates a cancel, save window
<dc>	Creates a discard, cancel, save window
<global>	Used for setting some settings
<dictionary>	Multiple language translations
<script>	TWimp script containing a list of actions

TWimp Keywords

This section contains a list of keywords that can be used to define items. Some keywords only apply to specific item types and will be ignored if the item does not support that keyword.

Keywords are not case sensitive. Hyphens are optional. Some have alternatives which are currently not listed. Position and Pos can both be used and do the same thing.

Where an X co-ordinate type is provided, there is usually a Y type. Most Y versions have been removed for clarity.

Descriptions that contain '****' are working but still could be changed in future versions.

Keyword	Description	Type
Text	Text of title of the item	S
Text-length	Maximum length of text that can be held (default=auto)	N
Visible-text	TWimp_Read only. Return text if icon not hidden or greyed	S
Sprite	Sprite of the item. Currently icons only	S
Sprite-length	Maximum name length of sprite that can be held (or auto)	S
Help	Help text for !Help to show	S
Pre	Label text to put to left of icon	S
Post	Label text to put to right of icon	S
Pre-length	Maximum length of pre text (or auto)	N
Post-length	Maximum length of post text (or auto)	N
Writable	Make the icon writable	
Allow	Characters that are accepted (Risc OS Format)	S
Positioning		
Pos	Position. Use as Pos: x,y,width,height	Pos
Width, Height	Position. Item width or item height	Pos
Auto-size	Adjust size automatically. For non-scrollbar windows	Pos
Pre-width, Post-width	Set icon pre/post label width. There is no height version	Pos
Pos-size	Position and size. Pos-size: x,y,width,height	Pos
Size	Position, sets width and height. Size: width,height	Pos
X	X position. From left	Pos
Y	Y position. From top	Pos
Add-x, Sub-x	Add or subtract value. Add-x : Number [,MaxOrMin]	N
Add-width/height	Add/Sub width or height of item with optional min, maximum	N
Top-left	Set from top left of window/screen. Top-left: x,y,width,height	Pos
Top-right	Set from top right window/screen. Top-right: x,y,width,height	Pos
Bottom-left	Set from bottom left wind/scn. Top-bottom: x,y,width,height	Pos
Bottom-right	Set from bottom right wind/scn. Bottom-right: x,y,width,height	Pos
Bottom-right-auto	Position is recalculated when window max extent is changed	Pos
Bottom-right-visible	Position is recalculated when window is moved	Pos
Max-x	Maximum x position. From left. For windows	Pos

Keyword	Description	Type
Max-y	Maximum y position. From top. For windows	Pos
Right-x	Maximum x position. From right. For windows	Pos
Bottom-y	Maximum y position. From bottom. For windows	Pos
Front	Bring window to the front	
Scroll-x	Scroll positions of a window.	Pos
Auto-scroll-x	Auto-scroll if icon dragged. Auto-scroll : left,top,right,bottom	D
No-auto-scroll-x	Turns off the auto-scroll	
Min-width	Window minimum width and height	N
Text-align	Parameter is left, right, left-top, left-center etc	D
Sprite-align	Sprite position inside the icon. Left-top, center, right etc	D
Text-gap, Text-gap-x	The margin between the icon border and the text	N
Sprite-gap, ...-gap-x	The margin between the icon border and the sprite	N
Positioning Extra		
Base-x	All icons made after this have x value added	N
Add-base-x	Add to base-x	N
Sub-base-x	Subtract from base-x	N
Grid	****	
Grid-x	****	
Grid-off	****	
Menu		
Gap	Gap between items	N
Width	Width of menu (overridden by Risc OS)	N
Height	Height of each item in the menu	N
Menu Items		
Sub, Sub-menu	Name of submenu or window	S
Writable	Writable item	
Tick	Tick the menu item	
No-tick, Swap-tick	Change status of tick in menu item	
Dotted	Dotted line after the menuitem	
No-dotted, swap-do...	Change status of dotted line after this menuitem	
Key-open	Opens the submenu if the shortcut key is press (see on-key)	
Colours		
Colour	Foreground colour number (Wimp), name or 3 or 6-digit hex	C
Bg-colour	Background colour	C
Title-colour	Window or menu title colour	C
Title-bg-colour	Window or menu title background colour	C
Title-highlight	Window title colour when it or an icon in it has the caret	C
Work-colour	Work area colour of window or menu	C
Work-bg-colour	Work background colour of window or menu	C
Scroll-colour	Scrollbar colour (Risc OS may override this)	C
Scroll-bg-colour	Scrollbar background colour (Risc OS may override this)	C
Pre-colour	Set the colour of the pre text, if defined	C
Post-colour	Set the colour of the post text, if defnind	C
No-border	Remove all window borders	
Border	Put window borders back	
Status		
Grey	Grey an icon or menu item	
No-Grey, Swap-grey	Change status of greyed item	
Select	Tick or select an icon or menu item	

Keyword	Description	Type
No-select	Change status of selection of item	
Swap-selected	Change status of selection of item	
No-border	Switch border off icon	
Swap-border	Swap border status of icon	
Border	On its own or with Border: line, in, out, ridge, channel, action, default, writable, ok, cancel for 3D borders. Add “,auto-size” for 3D borders to adjust co-ordinates.	D
Auto-grey	Takes a list of items which will be greyed automatically depending on if this item is selected or not. Put a negative symbol in front on item names to invert. Eg) Auto-grey : icon1,icon2,-icon3,-icon4	D
Window Furniture		
Has-title, No-title	Window title bar status	
Swap-title	Swaps if the window has a title bar	
Has-close, No-close	Window close button status, also swap-close	
Has-back, No-back	Also Swap-back	
Has-toggle, No-toggle	Also Swap-toggle	
Has-adjust, No-adjust	Also Swap-adjust	
No-scroll-x	No horizontal scroll bar. Will appear if window is too small	
Never-scroll-x	No horizontal scroll bar. TWimp will never add one	
Swap-scroll-x	Swap the horizontal scrollbar status	
No-scroll-bars	Switch all scrollbars off. Also has-scroll-bars to switch on	
Modal	Removes all window furniture except for the title bar	
Toolbar	Sets window furniture to make suitable for a toolbar	
Pane	Sets window furniture to make suitable for a pane window	
Title-is-filename	Window will act filer-like if title contains a filename	
Title-is-not-filename	Unset the filename setting for the window	
Modified, Not-modi...	Set or unset the asterisk. Alters title-is-filename actions	
Radio and Options		
Group	Group name (not number) for radio buttons	S
Frame	Frame name. Useful for finding extent of a group of icons	S
Set-group	Sets a group name. All icons made after this have this name	S
Unset-group	Stops following icons having the set-group name	
Set-frame	Also unset-frame	S
Multi-select	Radio button can be selected with adjust without cancelling	
Numbers		
Value	Set the value of a number icon	N
Min, Max	Set the minimum or maximum of the value in the icon	N
Step	Amount added to the value when the arrows are clicked	N
Speed	Amount added when arrows clicked. Can type other values	N
Has-arrows	Add arrows to the number icon	
No-arrows	Remove arrows from the number icons	
Continuous	Returns slider values continuous to program via on-change	
Pop-up		
Pop-up	Name of menu to open when the menu button clicked	S
Pop-up-list	List of strings to make a menu quickly. (“Title”, “Item 1”, ...)	S
Iconbar Position		
Iconbar	Left, right, Left-high, right-high, left-low, right-low accepted	D
Min-y, Max-y	Recommend to use for positioning iconbar items	N

Keyword	Description	Type
Multiline		
Multi-line	Make this icon a multi-line icon (optional vertical spacing)	[N]
Pre-multi-line	Make the pre or post icon multi-line (also Post-multi-line)	
Hide		
Hide, Hidden	Hide an icon	
Un-hide	Show the icon	
Other		
Caret	Caret in item (update). Always on window open (create)	
Allow-caret	Clicking the item will set the caret. Also allow-focus	
No-caret	Removes the caret if was in this item. Used when updating	
Pointer	Set pointer shape. Pointer: "Sprite",x,y	D
Freeze	Freezes a window so the user cannot move it	
No-freeze	Allow the user to move the window again.	
Swap-freeze	Swap a windows freeze status	
Half-sprite	Show sprite at half size (Also no-half-sprite and swap)	
Password	Icon is password entry and character entry is hidden	
Related	List of related items. Window can have <DCS> or <quit>. DCS and quit can be related to a <save>	S
Toolbars and Panes	These are defined as icons (<icon>toolbar: windname)	
Toolbar	Set window name that is used as a toolbar	S
Toolbar-left	Also toolbar-right, toolbar-top, toolbar-bottom	S
Pane, Subwindow	Window name used as a pane	
Pinboard		
Mini-text	Window title used by Pinboard to override default	S
Mini-sprite	Sprite name to use by Pinboard. Define in window or global	S

Any where type is included in the right hand column should usually have a colon after the keyword followed by the parameters. Types in square brackets are optional. Multi-line has an optional parameter and can be multi-line or multi-line:40.

Type	Type Description	Type Example
S	String/Text	Text: "Hello World"
N	Numerical	X: 100
Pos	Position	Pos: 100,150,100,50 Icon positions can use lastx, lasty, lastwidth, last height to copy the place of the last defined icon. Each of these can also have +X or -X after them where X is a whole number. Example Pos: lastx, lasty+100, lastwidth, lastheight Can refer to positions of specific icon using icon[<iconName>]x frame[<frameName>]x group[<groupName>]x pre[<iconName>]x post[<iconName>]x where x can be x,y,width,height,top,bottom,left,right Example Pos: frame[FR]right+12,frame[FR]y,100,100
(blank)	No parameter required	Has-title
A	Action	On-click: -open mainwindow
E	Event action	On-click: -event GREYICONS
D	Described	Described in the table above

TWimp Action Keywords

Action keywords tell TWimp what to do when the action happens. For example, an on-click action will run when the item is clicked on with select or adjust.

Please note that some combinations may not work as expected due to the way the Risc OS Wimp functions.

There are two types of action. Actions and Events.

An action can run and return instantly to your program with full details in TWimp_PollData.

An event can run and return to your program at a later time. TWimp_PollData information will be very limited.

Keyword	Run when	Type
On-click	Clicked with select or adjust mouse buttons	A
On-select	Clicked with select	A
On-menu	Clicked with menu	A
On-adjust	Clicked with adjust	A
On-d-click	Double clicked with select or adjust. Also on-double-click	A
On-d-select	Double clicked with select	A
On-d-adjust	Double clicked with adjust	A
On-drag	Dragged with select or adjust (Fixed size box drag)	A
On-drag-select	Dragged with select	A
On-drag-adjust	Dragged with adjust	A
On-hover	Mouse pointer is over the item	A
On-open	Item is opened (currently windows and menus only)	E
On-close	Item is closed (currently windows and menus only)	E
On-key-...	Key is pressed. On-key-cs-F4: CONTROLSHIFTF4	A*
On-change	An icons data or number has changed	A
On-enter	Mouse has entered the window	A
On-leave	Mouse has left the window	A
On-gain-caret	Window or icon in it has gained the caret	A
On-lose-caret	Window has lost the caret	A
On-load	A file dragged to this item (or double-clicked if in <global>)	A
On-save	Save should be performed on a <save> window	A
On-discard	Discard clicked on <DCS> and <quit> windows	A
On-cancel	Cancel clicked on <DCS> window	A

* on-key-... can accept cs, c or s before the keyname. F1-F11 plus Backspace, Return, Escape, Home, Delete, Print, Tab, End, Left, Right, Down, Up and Insert keys can be used. F12 can not be used. If used on a menuitem that menuitem will have the shortcut key inserted into the menu when it is shown. Menus should only have the key listed (on-key-cs-F7) and the on-click action will be run. Other items should have actions following (on-key-cs-F7 : -open

settings). Menu key shortcuts will run the on-click action for the menu item if the key is pressed while the caret is in a window or icon that would open the menu if the mouse menu button would open that menu.

Action definitions that start with a hyphen (-) signify that TWimp should attempt to do the action. Actions listed without the hyphen will be returned to your program. Only one can be returned to your program.

To open a window called "TheWindow" when clicking on an item you can use the following action in the item. Note that the open instruction starts with a hyphen to suggest TWimp will do this.

```
on-click : -open TheWindow
```

You may want more complicated actions that requires your program to do something in response to the click.

A method for returning to your program with some text in reason\$ could be as follows:

```
on-click : DoAction
```

In your program you would need to add something to handle this:

```
WHEN "DoAction" : SYS "TWimp_Error", "Item was clicked"
```

This will show an error message when the item containing that on-click instruction is clicked on.

Please note that not all items can respond to all actions. You may receive single click actions before a double click for some icons when the user double clicks.

Only one action name in reason\$ can be returned to your program with the full TWimp_PollData. This has all the information about what happened.

An instruction called -event can send multiple reasons back to your program and you can have a list of actions in your item.

```
on-click: -open MainWindow  
on-click: -event DoAction  
on-click: DoThisNow
```

All of the above will be performed. The window will be opened if it exists. Your program will also get DoThisNow in reason\$.

The DoAction event will be delivered when Risc OS and your program have a bit of free time. This is usually instantly but not guaranteed. Less data is available to you when an event is delivered. Events are delivered as fast as possible using the NULL poll feature in Risc OS. If you have multiple events they will allow other tasks to perform actions inbetween.

Available actions. ITEM* should be your item name. UPD** is an update string that can contain keywords as described in the keyword chapter.

Action	Does this
-quit	Sends QUIT to your program. Checks for windows that contain files and opens related <DCS> or <quit> window if related:window was set and file is marked as modified
-open ITEM	Opens an item. An icon or menuitem will open the window/menu
-close ITEM	Closes an item
-open-at ITEM "UPD"	Updates the item and opens it. Eg) -open-at Window "x:500 y:500"
-open-as-menu ITEM	Opens a window or menu item like a menu. Closes automatically
-update ITEM "UPD"	Updates an item. Eg) -update Icon "Text:'Hello' x:100 color:blue"
-update-selected	Same as update but only runs if this item is selected
-update-no-selected	Same as update but only runs if this item is not selected
-update-tick	Same as update but only runs if this item is ticked
-update-no-tick	Same as update but only runs if this item is not ticked
-save SAVEWINDOW	Will perform the on-save event or open the save window
-event NAME	NAME will be returned to your program when OS is not busy
-action ACTIONNAME	Runs an action list defined in an <script> block
-set-var NAME, VAL NAME	Sets a TWimp variable. See Variable chapter for more details NAME will be returned to your program. Only one per action

* More than one item can be sent as an item to operate on multiple items at the same time.

```
<button>text : "Grey" on-click : -update  
icon1,icon2,menuitem1 "swap-grey"
```

** Update strings can run on the current item if the item name is not included. You must surround your update string with quotes (of either quote type) so TWimp understands it is not the name of an item.

```
<button>text : "Button" on-click : -update "grey"
```

Example Windows

Most windows will be similar to each other. Here are some popular types.

```
// Standard window with size and title. Will have scrollbars.
<window mainwindow> text:"The title"
                      size:800,400

// Standard window without scrollbars but has close and back
// clicking menu will open item mainmenu if it exists
<window mainwindow> text:"The title"
                      size:800,400
                      no-scrollbars
                      on-menu:-open mainmenu

// Modal window. Only has the title bar, useful for menus
<window fileinfo>    modal
                      text:"File information"
                      size:600,300

// Windows without scrollbars can be easily autosized
// Moving, creating or deleting icons will adjust the size
// automatically. Try changing the position of the OK button
// and the window will change size
<window test>    no-scroll-bars
                  text:"Title"
                  auto-size
<ok> pos:500,500

// Window with 24-bit colours, on-close will send QUIT
// message to your program, help and clicking will make the
// window taller. Allows the caret if the window is clicked
// and responds to F4 key.
<window mainwindow>
    text:"The title"
    no-scrollbars
    colour:red
    bg-colour:#99AABB
    title-bg-colour:green
    on-close:QUIT
    help:"The main window"
    on-click:-update mainwindow "add-height:400"
    allow-caret
    on-key-F4:-update mainwindow
                  "sub-height:400,400"
```

Example Icons

Here is a list of example icons that can be created.

```
// Make an iconbar icon that opens mainwindow when clicked
// with select, adjust closes and menu opens iconbarmenu
<iconbar>    sprite:file_fff text:"TheIcon"
              on-select: -open mainwindow
              on-adjust: -close mainwindow
              on-menu: -open iconbarmenu

// Simple icon called the_icon, positioned
<icon the_icon> text:"Hello" x:50 y:50

// Simple icon called the_icon, positioned & width & height
<icon the_icon> text:"Hello" x:50 y:50 width:400
height:52

// Simple icon called the_icon, using pos
<icon the_icon> text:"Hello" pos:50,50,400,52

// The same icon with colours, 3D border, writable and help
<icon the_icon> text:"Hello" pos:50,50,400,52
                  colour:yellow bg-colour:red
                  border:in
                  writable
                  help:"This is a demo icon"
```

There are lots of predefined icons that you can use.

```
// Display icon, will be 3D in grey colour and has pre label
// can use <display> or <data> to define
<display the_text> text:"The text goes here" pos:50,50
                  pre:"Data"

// Button icon. Simple and clickable with on-click
<button> text:"Close" on-click:-close mainwindow

// Repeater icon. Will return to your program over and over
// if the mouse button is held down over the icon
<repeater> text:"Add one"
            on-select:ADDONE
            on-adjust:SUBONE
```

```

// OK button. If one defined for the window and enter is
// pressed, the on-click or on-select action will be run if
// the window has the caret. Close mainw and return to prog.
<ok> text:"Ok" pos:800,800    on-click:-close mainw
                                on-click:OKPRESSED

// Cancel button. If one defined for the window and escape is
// pressed, the on-click or on-select action will be run if
// the window has the caret
<cancel> text:"Cancel"    pos:800,800
                                on-click:-close mainwindow

// Writable icon, defining maximum number of characters
// and also has a pre label and traps a keypress returning
// escape to your program
<writable firstname>    pos:200,20
                                text:"" text-length:255
                                pre: "First name"
                                on-key-escape:ESCPRESSED

// Option button that will automatically grey or ungrey
// list of icons, groups or frame names depending on option
// selected status and the + or - in front of the namelist
<option the_opt>    pos:20,20
                                text:"Option text"
                                auto-grey: +icon1,-icon2

// Radio button with a group name, only one in the group can
// be selected and one will be selected at start
<radio icon1>    pos:20,80
                                text: "Radio text"
                                group: RADIO_GROUP

// Number icon allows a whole number to be entered
// and also has pre and post text for the icon
// Returns CHANGED to your program when updated
<number number>    pos:500,500
                                value:0 min:-100 max:100
                                has-arrows
                                pre: "Before" post: "After"
                                on-change:CHANGED

```

```

// Make a slider icon, can also use <sliderv> for vertical
// Returns SLIDERCHANGE to your program continuously while
// dragging because continuous is set
<slider volume>    pos:250,250
                   value:0 min:-100 max:100
                   has-arrows has-number
                   pre:"Volume"
                   on-change:SLIDERCHANGE
                   continuous

// Create a basic label icon
<label> text:"Label" pos:150,150

// Pop-up icon attached to a quick menu. Define width but not
// the height so will be default height
<popup pu>    pos:500,20,300
              pop-up-data:"Title","Jan","Feb","Mar"

// Pop-up icon attached to a TWimp <menu> item called themenu
<popup pu>    pos:500,100,300
              pop-up:themenu

```


Example Menus

Menus can be made in a very similar format to windows and icons. Menus and items will be listed together here.

```
// Create a main menu called ibmenu with a quit option
// that returns QUIT to your program when selected
<menu ibmenu> text:"MyApp"
<item> text:"Quit" on-click:"QUIT"

// Menu with a submenu pointing to an about this program
// window, also defined below
<menu ibmenu> text:"MyApp"
<item> text:"Info" submenu:aboutw
<item> text:"Quit" on-click:"QUIT"

<about aboutw>purpose:"Demo" author:"Me" version:"0.01"

// Add a save window to this menu, requires the <save> line
// Adds on-key-F3 which is semi-automated when attached to a
// window that has the caret and on-menu in the window opens
// this menu
<menu ibmenu> text:"MyApp"
<item> text:"Info" submenu:aboutw
<item> text:"Save" submenu:savew on-key-F3
<item> text:"Quit" on-click:"QUIT"

<about aboutw>purpose:"Demo" author:"Me" version:"0.01"
<save savew>filename:"Unknown" sprite:file_ff9

// Make two menus and have one as the submenu. They can be
// made in any order
<menu mainmenu>      text:"Main Menu"
<item>               text:"Help"
<item>               text:"Filetype" submenu:filetypes
<item>               text:"Other"

<menu filetypes>     text:"Filetypes"
<item>               text:"Sprite"
<item>               text:"Module"
<item>               text:"Text"
```

Opening and Closing Items

To open or close an item from a TWimp item definition an action can be created. This takes the action type such as on-click and the action to perform following.

```
// Iconbar that opens something called mainwindow when select
// is pressed, closing it when adjust is pressed and opening
// ibmenu when the menu button is pressed
<iconbar ib>    sprite:file_fff
                on-select:      -open mainwindow
                on-adjust:      -close mainwindow
                on-menu:         -open ibmenu
```

To open and position at the same time openat has also been defined which can take a position string as the second parameter.

```
// Opens mainwindow at 50,50 when clicked
<icon>    text:"Open"
          on-click : -openat mainwindow "x:50 y:50"
```

SWI calls for opening and closing are also defined as follows.

TWimp_Open

Entry: R0 Item name

TWimp_OpenAt

Entry: R0 Item name
 R1 Position string

TWimp_Close

Entry: R0 Item name

Most of the time, using TWimp_Open should work well and menu positions are calculated to be opened in a sensible place even when using the SWI call by checking what the last action the user took.

An item name can be an icon or menu item name and TWimp will open the relevant window or menu. This means by using icon or menu item names in open or close, you will find it easier to move icons to different windows as your programs get larger and you may decide to split one window with many options into two.

Updating Items from your Program

You will likely want to update icons or read back data from them in your program. Some simple SWI calls are provided for this.

TWimp_Update

Entry: R0 Item name, list of item names, group names or frame names
 R1 Update string. Same format as the file described above

REM Update a sprite example

```
SYS "TWimp_Update","iconbaricon","sprite:switcher"
```

REM Update multiple icons plus everything marked as frame:nameframe to be greyed out

```
SYS "TWimp_Update","firstname surname nameframe","grey"
```

This SWI can be a little awkward if you want to update text because the update string will be parsed. A nicer SWI can be used to update text or sprites.

TWimp_UpdateText

TWimp_UpdateSprite

Entry: R0 Item name or list of names
 R1 Updated text or spritename

REM Update text to say "grey"

REM Note if using TWimp_Update, this would grey the icon!

```
SYS "TWimp_UpdateText","icon","grey"
```

Some similar SWIs can be used for numbers or values so you do not have to convert them to strings for TWimp_Update.

TWimp_UpdateValue

Entry: R0 Item name or list of names
 R1 Value (numeric)

TWimp_UpdateValue will update the 'value' of an item, such as a number icon or slider. To update any other numeric value like icon positions, UpdateNumber can be used.

TWimp_UpdateNumber

Entry: R0 Item name or list of names
 R1 Thing to update with a number (example: width)
 R2 Value to update with (numeric)

Reading Item Data from your Program

There are some similar SWI calls for reading item data back to your program.

TWimp_Read

Entry: R0 Item name
 R1 List of items to read
Exit: R0 Value or string
 R1 Value or string

 R7 Value or string

```
REM Read in x, y, width, height and text of an icon
SYS "TWimp_Read","iconname","x y width height text" TO
x%,y%,w%,h%,text$
```

```
REM Read in text from iconname but only if the icon
REM is not hidden and if the icon is not grey
REM If hidden or grey text$ will be an empty string
SYS "TWimp_Read","iconname","visible-text" TO text$
```

For consistency and convenience, the following SWI calls are also available.

TWimp_ReadText

TWimp_ReadSprite

TWimp_ReadValue

Entry: R0 Item name
Exit: R0 Text string, sprite string or value

Unlike the update SWI and actions, the read versions will create an error if the parameter is not recognised. This is to ensure your program will not continue with potentially bad results. You can use the X form of the SWI to suppress or handle these errors as you see fit.

A useful tool is to read only the visible-text of an icon. This will normally return the text in the icon as normal. If the icon is hidden or grey then it will return an empty string.

```
REM text$ will be empty string if the icon is grey or hidden
SYS "TWimp_Read","icon","visible-text" TO text$
```

Reading Data from the Wimp

When your program is entered from the Wimp, you will have a reason string. There is a lot more data you can access from TWimp such as mouse positions, windows, icons, key presses and much more.

TWimp_PollData

Entry: R0 List of poll data item you would like to read
Exit: R0 First poll data item value or string
R1 Second poll data item value or string
...
R7 Eighth poll data item value or string

REM In response to an on-load or on-save event:

SYS "TWimp_PollData","filename" TO filename\$

REM Which window name and icon name were clicked?

SYS "TWimp_PollData","window icon" TO window\$,icon\$

REM Read where the mouse pointer is inside the window

REM X position is from the left of the window extent

REM Y position is from the top and is a positive value

SYS "TWimp_PollData","wind-x wind-y" TO x%,y%

The wind-x and wind-y values will automatically convert the screen position to the window position, taking into account where the window is, scrollbar positions and even calculating the zoom level of the window. This means you no longer have to convert these values yourself. If you create an icon at 100,100 and the user clicks on the top-left pixel of that icon, wind-x and wind-y will both contain 100 regardless of where the window is on the screen, how the scrollbars are set or what the zoom level is.

Some of the values that can read are in the table below. They are not case sensitive. Where X values are shown, Y values are usually available but are not shown for clarity.

Some values may vary or not be available depending on the reason your program was called. The 'X' parameter may be the mouse position on the screen when a mouse click has been returned but the window position during a window open event. There are some specific calls to help with this such as mouse-x to return the screen position of the mouse and window-x to return where the event took place within a windows co-ordinates.

Most values have a default value when they do not exist. Example may be trying to read the window name in a timer event. It will return an empty string.

These values are stored at the time of the Wimp poll loop. You can read PollData to see if the shift key was pressed at the time of the poll loop rather than reading it now, which may be different if the user released it quicker than your program reacts if you are doing something task intensive.

PollData	Returns this value
General	
Window	Window name
Icon	Icon name
Menu	Menu name
Menu-item	Menu item name
Menu-window	Window name where menu was opened
Menu-icon	Window name where menu was opened
Draw	Draw item name
Task-name	Name of the task as you provided by TWimp_Start
Zero	Set the output register to zero
Preserve	Skip this register
X	X position
Wind-x, Window-x	X position but within the window
Minx, Maxx	Minimum and maximum x positions (window open event)
Missed	Number of timer events missed (timer events only)
Window	
Scroll-x	The X position of the window scrollbar
Scroll-x-dir	The scrolling direction that a wimp scroll event has requested
Behind	Window handle of the window this is behind*
Menu	
Menu-x	The x on the screen of where the open menu was opened
Keypresses	
Char, character	Character code of the key pressed (key press events)
Left-shift, right-shift	Was a specific shift key pressed?
Shift	Was any shift key pressed?
Ctrl, left-ctrl, right-ctrl	Was control pressed?
Alt, left-alt, right-alt	Was an alt key pressed?
Left-Risc-OS	Was the left Risc OS key pressed?
Right-Risc-OS	Was the right Risc OS key pressed?
Risc-OS-menu	Was the menu key pressed?
Escape	Was the escape key pressed?
Caret	
Caret-window	Window name where caret is
Caret-icon	Icon name where caret is
Caret-x	The caret position
Caret-height, height	The caret height
Caret-index, index	The index position of the caret into an icons text
Drags	
Drag-start-window	Window name where drag started
Drag-start-icon	Icon name where drag started
Drag-end-window	Window name where drag ended
Drag-end-icon	Icon name where drag ended
Drag-start-wind-x	Window co-ordinates of where the drag started
Drag-end-wind-x	Window co-ordinates of where the drag end

PollData

Drag-start-x
 Drag-start-min-x
 Drag-end-x
 Drag-end-min-x
 Drag-length-x

Mouse

Mouse-x
 Mouse-b, mouse-buttons
 Mouse-select, -left

Mouse-drag-select
 Mouse-drag
 Mouse-double-select
 Mouse-double

Load/Save

Filename
 Filetype
 Filesize
 Col, column, row
 Loadtype
 Saving-window

Messages

Message-length, msg-len
 Sender
 My-ref
 Your-ref
 Message-reason, message
 Prequit
 Dest-window
 Dest-icon

Poll Information#

Poll-reason
 Poll-number
 Poll-word
 Poll-address, poll-addr
 Wimp-block

Handles*

Window-handle
 Icon-handle
 Menu-handle
 Menu-opened-window-handle
 Menu-opened-icon-handle

 Caret-window-handle
 Caret-icon-handle
 Drag-start-window-handle
 Drag-end-window-handle
 Drag-start-icon-handle

Returns this value

The drag starting position, screen units
 The minimum box position of the start of a drag. Also max, y
 The drag ending position, screen units
 The minimum box position of the end of a drag. Also max, y
 The length of the drag from start to end in the x dimension

 The mouse X position on the screen
 The mouse buttons pressed
 Was the select mouse button pressed? Also menu, adjust, middle, right
 Is the event due to a drag with the select button?
 Is the event due to any type of drag?
 Is the event due to the mouse select button double clicked?
 Is the event due to any mouse button double clicked?

 Filename of file to load or save as
 Filetype of file when loading
 Estimate filesize as defined by Risc OS
 Column or row number when multiple files loading (or -1)
 Will be DRAGGED or DOUBLE if file was dragged/dbl clicked
 Window linked to the last save that has title-is-a-file set

 Length of the Wimp message sent
 Task handle of message sender
 My-ref as defined by the Wimp
 Your-ref as defined by the Wimp
 The message number as defined by the Wimp
 Set if quit is just you, otherwise shutting down.
 Destination window handle on a load message*
 Destination window handle on a load message*

 The reason text that your program was called with#
 The Wimp number code (reason code in Wimp_Poll)#
 The pollword value#
 The pollword address#
 The wimp block pointer that was used#

 The window handle assigned by the Wimp*
 The icon handle assigned by TWimp*
 The menu pointer assigned by TWimp*
 The window handle of the window where the last menu was opened*
 The icon handle of the window where the last menu was opened*

 The window handle that has the caret*
 The icon handle that has the caret*
 The starting window handle where a drag happened*
 The ending window handle where a drag ended*
 The starting icon handle where a drag happened*

PollData

Drag-end-icon-handle

Returns this value

The ending icon handle where a drag ended*

Events

Event-data

The string parameter from -event or TWimp_Event

* Handles should not be used if possible. You should use the same polldata item without the '-handle' on the end to get the name of the item instead. TWimp reserves the right to renumber any handles during runtime. This usually occurs during any type of TWimp_Update call when it needs to but can occur during other TWimp SWI calls.

Poll information can be used like the standard wimp. You can read from the Wimp block but not write to it on some versions of Risc OS. Note that any handles in these blocks (window handles, icon handles) can be updated on any call to Wimp_Update or similar and you should be aware of this.

Groups and Frames

Groups and frames can be set for all icons and menu items.

Radio buttons and menu selections are the most useful items to have group names. TWimp will ensure that only one is selected as well as ensuring one is selected in each group.

```
// Radio buttons with group names
<radio max> text:"Max" group:volume pos:50,50
<radio min> text:"Min" group:volume pos:50,110
<radio off> text:"Off" group:volume pos:50,170
```

The above definitions will create three radio buttons. They are all in a group called volume so one will be selected by default. This will be the first one if you have not set one to be selected with the keyword select or selected.

To define a group set, it is usually easier in text definition files to use set-group and unset-group.

```
// Radio buttons with group names
set-group:volume
<radio max> text:"Max" pos:50,50
<radio min> text:"Min" pos:50,110
<radio off> text:"Off" pos:50,170
unset-group
```

To read which has been selected in your program use TWimp_Read for each icon. A better approach can be using TWimp_ReadGroup

```
// Read which icon name is selected
SYS "TWimp_ReadGroup","volume" TO selected$
```

The string returned will be the first radio button selected in the volume group and will be the name of the icon or menu item.

Frame names are similar to group names but are used for positioning. Use frame:<name> to set a frame name on an icon or set-frame:<name> unset-frame around a group of icons. These can be used to find the extent and block size of a group of icons.

If you define a frame name and also a <frame> icon with the same name, the frame icon will be auto-positioned around the icons for you.

```
// Make a frame around an icon with no positioning for
// the frame itself
set-frame:volumeframe
<frame volumeframe> text:"Volume"
set-group:volumegroup
```

```
<radio max> text:"Max" pos:50,50  
<radio min> text:"Min" pos:50,110  
<radio off> text:"Off" pos:50,170  
unset-group  
unset-frame
```

Frames can be useful for positioning other icons around a group of icons by using the positions. For example, you can start a new icon to the right of all of the icons in the above code even if the sizes change or new icons are added with different sizes. See the chapter on advanced icon positioning for more details.

Timer Events

TWimp allows you to create timer events. These operate by using the standard Wimp poll null codes which means you are not guaranteed to get the event but you will get them as close as possible.

TWimp_AddTimer

Entry: R0 Text to return to your program
 R1 Number of centi-seconds (100 = 1 second)

This will add a timer event to your program. The text you provided will be returned to your program after the timer amount and then every timer amount after that or as close as the Wimp can manage.

The standard timer will return its first call after the number of centi-seconds have passed.

To return the value the text value to your program now as well as every timer amount after that, simply set the number of centi-seconds to the negative amount.

Multiple timers can be created. Only one timer event with a specific name can be made. Calling TWimp_AddTimer with the same name as a previous timer event will replace the previous timer.

REM Create a timer to return every 60 seconds

```
SYS "TWimp_AddTimer","MINUTE",6000
```

REM Create a timer to return now and then every 10 seconds

REM by using a negative time interval

```
SYS "TWimp_AddTimer","DOSOMETHING",-1000
```

TWimp will manage the timer for you to get as close as possible to the requested time intervals.

It is possible to miss events if another task uses system resources or a single tasking program takes over, such as pressing F12 to enter the command line.

To find the number of missed events, you can use TWimp_PollData to get the number of events that have been missed.

REM Did we miss any timer events? How many?

```
SYS "TWimp_PollData","missed" TO missed%
```

A counter would add one for this event and add the number of missed events to keep the counter in sync with real time.

Removing a timer is as simple as calling TWimp_RemoveTimer with the same text in R0 that was supplied in TWimp_AddTimer.

TWimp_RemoveTimer

Entry: R0 Text that was being returned to your program

Saving and Loading Files

The saving and loading of files has been simplified with TWimp handling most of the process.

Loading Files

All windows and icons, including iconbars can have the on-load event added to them which will let your program know when a load has happened by the user dragging a file onto that window or icon.

```
// Return LOAD to your program when any file is dragged
// onto this icon
<display> text:'Hello' on-load:LOAD
```

For specific file types, you can add the hexadecimal filetype on the end of the on-load command to filter the types. Multiple types are permitted for the same item, allowing TWimp to filter different types to different paths in your program. Directories are 1000 and applications 2000.

```
// Return SPRITE if a spritefile dragged here
// Return TEXT if a spritefile dragged here
<icon> text:'Drag'
        on-load-ff9:SPRITE
        on-load-fff:TEXT
```

To respond to a load event, you will need to have some code or procedures to handle the LOAD, SPRITE and TEXT reasons sent back to your program (or what you those to called them).

The simplest way to find out the filename is by using the PollData. You should also call the Loaded SWI once the file has been loaded so TWimp and the Wimp can handle the behind-the-scenes details.

```
REM Get the filename
SYS "TWimp_PollData","filename" TO filename$

REM Load your file here
REM Tell the Wimp your file loaded successfully
SYS "TWimp_Loaded"
```

There are a few other helpful things that can help your loading procedure. PollData can return loadtype which will return the string DOUBLE or DRAGGED and that will let you know if the file loading was dragged or double-clicked in a filer window. With an icon or window, this will always be the DRAGGED string returned to your program.

TWimp_Loaded can take an extra parameter. This is a list of windows which can be updated automatically for you. If a window is a standard window, the titlebar will change to the filename and adjust-clicking the close icon will now perform standard Risc OS operations for file windows. Any window listed that was created as a <save> window will have the filename updated instead of the titlebar.

REM Update the mainwindow and savewindow filenames

```
SYS "TWimp_Loaded", "mainwindow savewindow"
```

TWimp_Loaded

Entry: R0 List of windows to update with filename

Loading Files from Filer Double-Clicks

To capture when a user double-clicks a file in a filer window you simply put the event in a <global> section of your TWimp file.

```
// This will capture double-clicked sprites and text files
<global>
    on-load-ff9:SPRITE
    on-load-fff:TEXT
```

Note that when a file is double-clicked, PollData will return an empty string for the window and icon and will return DOUBLE if you ask for 'loadtype'.

REM Where and how was this loaded

```
SYS "TWimp_PollData", "filename window icon loadtype" TO
    filename$, window$, icon$, loadtype$
```

You must use TWimp_Loaded when a file has been double-clicked or the Wimp will ask other tasks if it would like to load the file. It is strongly advised that you always use TWimp_Loaded on every loaded file.

Loading a File When Your Application is Not Loaded

Files can be loaded into your applications automatically by TWimp but you will need to pass the filename along. First, follow the Loading Files from Filer Double-Clicks section above if you have not done this.

You will need to create a RunType variable to point to your applications !Run file. Using file type 007 and an application variable of My\$Dir, your !Boot file should contain the following line.

```
Set Alias$@RunType_007 <My$Dir>.!Run -Load %%*0
```

It is important that the -Load <filename> part of the command is included for TWimp to understand which part of the command is the filename. A double % is

used inside an Obey file to stop it being translated into a filename during the initial setup.

The !Run file should also be modified to pass your filename to your program and TWimp.

```
Run <My$Dir>.!RunImage %*0
```

No other code is required by your program. The load function will be called to your program on a null event from the Wimp.

Saving Files

Saving files is a similar simple interface. It is recommended that you use the <save> window item to keep the process simple.

```
// Create a save window
<save savewindow>    sprite:'file_ff9'
                     filename:'SpriteFile'
                     on-save:SAVETHEFILE
```

This should be linked to a menu item and ideally with the F3 key assigned to the menu item. The following should be made after a <menu> definition.

```
// Making a menu item
<item> text:'Save' submenu:savewindow on-key-F3
```

Any window that has this menu linked will automatically be able to handle the keypress for you, if the window or an icon in the window has the caret. This is linked by using 'on-menu: -open themenu' in the window definition. Something you have most likely done already, making this very convenient. To allow the caret in your window add a writable icon or use allow-caret in the window definition which allows a mouse click anywhere in the window to accept the caret.

When the user has dragged the icon to a filer window, the OK button in the save window is pressed, the user presses enter in the filename or the user clicks on the Save menu item you created, the save may begin.

In cases where the filename has not been completed the save window may appear or an error may occur informing the user to drag the icon to a filer window. What happens will depend on what the user did.

Where the filename is acceptable, the on-save event will be run. In this case returning SAVETHEFILE to your program. Most of the program is the same, except you call TWimp_Saved at the end.

```
REM Responding to SAVETHEFILE
SYS "TWimp_PollData","filename" TO filename$

REM Save the file to filename$
```

```
REM Set the filetype to the correct type
REM Tell TWimp that was have saved this file
SYS "TWimp_Saved","savewindow filewindow"
```

Note that TWimp_Saved can also take a list of windows. This will update relevant parts of those windows and mark the window as not-modified.

You should see that the load and save protocols in TWimp are very similar to each other.

TWimp_Saved

Entry: R0 List of windows to update with filename

Save Window Options and Radio Buttons

Save windows can have simple <option> or <radio> buttons in them and TWimp will adjust your window size automatically.

```
<save window_name> filetype:FF9 on-save:SAVE
<option save_selection> text:"Selection"
```

Note that the position of the option or radio button in a save window can vary depending on the version of the Wimp that is running. The new style save window has a cancel button and the old style only has the OK button.

Save Window Style

A window can be created using either style by creating it with <saveold> or <savenew> instead of <save>.

Note that your choice may be overridden by the user by setting the system variable TWimp\$SaveType to 1 for the old style or 2 for the new style. You must never set this variable in any distributed software. Use <savenew> or <saveold> if you have a preference or <save> to allow TWimp to choose. The user choice will override all of these.

Setting Windows to be File Windows

Windows can be updated using TWimp_Update to be file handling windows. If the title of the window is a filename, you can use title-is-file to tell TWimp that information. Then you can update the modified or not-modified status. Note that TWimp_Loaded and TWimp_Saved will alter these settings as it sees fit, as long as you included the window name in the list of names.

```
REM Tell TWimp that wind is a non-modified filename
SYS "TWimp_Update","wind","title-is-file not-modified"
```

By telling TWimp that your window is a filename, it will handle lots of standard Risc OS features for you that you would normally have to program yourself.

This includes using adjust when closing the window which should both close the window and open the filer window that contains the file. Holding shift and using adjust will keep the window open but still open the filer window. This is all done by setting the title of a window to a filename and setting 'title-is-file' for the window. Using 'modified' and 'not-modified' can change some of these automatic functions especially if your window has a discard, cancel, save window <DCS> or quit window <Quit> listed as 'related'.

```
// This window has a DCS window linked that will be used
// if the filename has been set in the title and is marked
// as modified
```

```
<window mainwindow>
    title-is-file
    modified
    text: "RAM:$.Test"
    related: ask_to_save
```

```
<dcs ask_to_save>
    on-discard: -update mainwindow "not-modified"
    on-discard: QUIT
    related: savewindow
```

```
<save savewindow>
    on-save: SAVE
```

The above definitions will stop you from closing mainwindow. Instead, the DCS window called ask_to_save will be opened since it is marked as related to mainwindow.

Clicking save will open the savewindow item which is marked as related to the DCS window.

Discard will quit the program by sending QUIT to the program. Note that it also marks the mainwindow as not-modified first so that quitting and closing this window will not reopen the DCS window again. ****

Opening and Closing Files

TWimp can open and close files using a TWimp_MiscOp SWI call. The advantage to using these calls instead of the OS is that TWimp will close and set the type of your file should your program crash.

Use open-w to create a file or setting an existing file to zero length ready to be written to. Open-r will open a file for reading. Close will close a file previously opened by TWimp. You must always use TWimp to close any files opened by TWimp. If your program ends unexpectedly or you forget to close the file, TWimp will do this when the program ends.

TWimp_MiscOp. “Open-w” and “Open-r”

Entry:	R0	Pointer to string “open-w” or “open-r” or “close”
	R1	Pointer to filename string
	R2	Filetype if writing to a file
Exit:	R0	File handle

TWimp_MiscOp, “Close”

Entry:	R0	Pointer to string “close”
	R1	Filehandle that was provided by TWimp

Saving and Loading Options

TWimp can load and save the options of windows and menus into a file in the users choices. This can be done with two simple SWI calls.

TWimp_SaveOptions

Entry: R0 Options filename (leafname only)
 R1 List of windows and menus that have the options

TWimp_LoadOptions

Entry: R0 Options filename (leafname only)

These SWI calls will load or save the status of all the icons or menu items in the list of windows or menus provided. The list can include individual icon or menu item names if required.

This will save:

- Selected status of option icons and radio icons
- Ticked status of menu items
- Greyed out status of all icons
- Hidden status of all icons
- Text of writable items
- Value of number or slider icons

Only the leaf name is required for saving a file and your application name will be used to create a directory in the users choices directory. You can save the status of more than one window or menu at a time. Using different filenames allows you to save different settings into different files.

```
REM Load options file. Put after windows & menus created
REM Usually after the TWimp_Load or TWimp_Create commands
SYS "TWimp_LoadOptions","settings"

REM Save options file
REM Saves all volume video & conf windows or menu data
SYS "TWimp_SaveOptions","settings","volume video conf"
```

Dragging Things (Fixed Sized Box)

Icons can be dragged in TWimp by setting the on-drag, on-drag-select or on-drag-adjust actions. The on-drag will happen if the user drags with either the select or adjust mouse buttons. Actions that you list will only happen when the drag ends. These are fixed size box drags and will be a dotted outline unless your icon contains a sprite where it may be the sprite that is dragged, depending on a number of factors.

```
// This will create a dotted outline when dragged
```

```
<icon DragIcon>      text:Hello World
                      pos:50,50
                      colour:white
                      bg-colour:blue
                      on-drag:DRAG
```

The above definition will create an icon that will allow a drag to happen and will return the string DRAG to your program when the drag is ended. You can read information using TWimp_PollData to get information about the drag start and end positions when it send this message to your program.

The following code will give an idea of how to read in the drag data and move and icon into the newly dragged position.

```
REM First read in start and end window names
SYS "TWimp_PollData","drag-start-window drag-end-
window" TO startw$,endw$

REM Check that the start and end were in the same window
IF startw$=endw$ THEN

REM Read in the window positions of the drag end
REM drag-end-wind-x gives the position within the window
REM drag-end-x would give the screen position
SYS "TWimp_PollData","drag-end-wind-x drag-end-wind-y"
TO endx$,endy%

REM Read in which icon was dragged
REM drag-end-icon would give the icon you dragged it over
REM drag-start-icon gives you the icon that was dragged
SYS "TWimp_PollData","drag-start-icon" TO starti$

REM Update the position of the icon dragged, using top-left
SYS "TWimp_Update",starti$,"top-left:"+STR$endx%
+", "+STR$endy%

ENDIF
```

There are many parameters that you can read in from PollData and they are listed in the chapter Reading Data from the Wimp.

Using drag-end-wind-x will give you the co-ordinates in the window, taking account of the position of the window, scrollbars and zoom levels. This is the co-ordinate from the top left of the window for the x position and the co-ordinate down from the top for the y position, the same as icon positions.

The Wimp can handle auto-scroll and TWimp supports this too. Defining a window with auto-scroll, auto-scroll-x or auto-scroll-y will cause a window to scroll automatically if you hover towards the end of a window that can be scrolled.

REM Create a window with auto-scrolling

```
<window MainWindow> title:"Main"  
                    auto-scroll
```

Events

Events are similar to actions and you can trigger these within your program. The technical differences between an event and an action are:

- Events are called on via null Wimp_Poll and will not be performed straight away. Other tasks may be switched to before your event is called
- Limited data is available from TWimp_PollData
- Multiple events can be queued up, only one action per poll loop will be called
- They can be used for non-urgent programming since they may not be called immediately
- Events will be called in the order you request them to be called

In order to create an event, either use the definition actions -event command or call TWimp_Event.

```
// Make two events happen if this icon is clicked
<ok>      on-click : -event EV_ONE
          on-click : -event EV_TWO
```

Both of the events above will be called. To respond to these events your program will receive the EV_ONE and EV_TWO strings in reason\$ in your poll loop as fast as the Wimp and TWimp can.

```
REM Call an event from your program
SYS "TWimp_Event","EV_ONE"
SYS "TWimp_Event","EV_TWO"
```

TWimp_Event

Entry:	R0	Event name string
	R1	Optional parameter or 0 if none

The above will do the same and queue up two events for your program to respond to.

Events can be used for repeatedly calling code instead of a timer, calling the same event again if you have not finished the work being done while giving other tasks a chance to multitask.

They can also be used for less urgent programming that needs to be done soon but does not have to be done right now and gives other tasks a chance to have a turn at running.

Events with Parameters

Events can be added that send a string parameter back to your program.

```
// Send data back to your program
<ok>      on-click : -event EVENTNAME,"Parameter"
```

This can also be achieved with the SWI.

```
REM Call an event with text data
SYS "TWimp_Event","EVENTNAME","Parameter"
```

The parameter is a string value. Sending a number will be in a string format. To read this data in your program, use TWimp_PollData.

```
REM Read the event parameter
SYS "TWimp_PollData","event-data" TO e$
```

Dynamically Creating and Deleting TWimp Items

You do not have to create items in an external file and can create or delete items while your program is running.

To create an item, call TWimp_Create with the creation string.

REM Make a window with one icon

```
SYS "TWimp_Create","<window mainwindow> title:'Main' "  
SYS "TWimp_Create","<option mute -in mainwindow>  
text:'Mute' "
```

Strings can be surrounded by single or double quotation marks so we have used single quotation marks in the above example to not clash with the BASIC string ending.

Creating the icon has a new option added. This is the -in and tells TWimp which window or menu something should be created in. You do not need this in a TWimp file or if the window and option are defined in the same TWimp_Create call. You do need to use it when creating an icon or menu item when on its own. This is to protect you from accidentality putting a new create window between those lines and now your icon is being made in the wrong or an unexpected window. By making you explicitly state where it should be created, there is no confusion. An error will be made if you miss the -in from the line if it is required.

TWimp_Create

Entry:	R0	Creation string, same as file input but with -in required
	R1-R9	Replacement values

Create can be used to make multiple versions of the same items easier. TWimp will replace %1 with the value text pointed to by R1. You can also use %1 to %9 which will replace using the text pointed to by the matching register number. Using %% will replace that with a single % character.

**REM Make a window with the name Main_1 or Main_2 or any other
REM number depending on the value of wind%**

```
SYS "TWimp_Create",  
    "<window Main_%1 text:'Main' ", STR$wind%
```

STR\$wind% will be appended to the end of Main_ in the definition.

You can also use variables in the string. See the chapter on variables.

When working in some programming languages there is a limit to the length of a string or line length. This can often be overcome by using Create followed by Update.

REM Create a window, setting the title and on-click by

REM using TWimp_Update on separate lines

```
SYS "TWimp_Create","<window mainwindow> size:400,400"
```

```
SYS "TWimp_Update","mainwindow","text:'The title here'"
```

```
SYS "TWimp_Update","mainwindow","on-click:-error Hello"
```

Variables

TWimp allows variables in many places. They allow you to create items dynamically with far more ease than making your own strings within many programming languages and also can make your code more readable.

Currently, variables are case sensitive although this may change in the future. You must only use characters A-Z, a-z, 0-9 or the underscore character. Even if other characters currently work this may change. Spaces must not be used around the variable name when setting or changing a variable.

```
REM Create two variables
```

```
REM This one will be deleted on the next Wimp_Poll loop
```

```
SYS "TWimp_MiscOp","add-var","window","settings_window"
```

```
REM This one is permanent
```

```
SYS "TWimp_MiscOp","add-var-perm","var_name","kept"
```

Variables must be text strings. If you wish to use a number, it will need to be converted into the correct format into a string.

TWimp_MiscOp, "Add-var"

Entry:	R0	Pointer to string "add-var", "set-var"
	R1	Pointer to name of variable
	R2	Pointer to string of variable value

This SWI will create a permanent variable that will be deleted only when you request the variable to be deleted.

TWimp_MiscOp, "Add-var-tmp"

Entry:	R0	Pointer to string "add-var-tmp", "set-var-tmp"
	R1	Pointer to name of variable
	R2	Pointer to string of variable value

This SWI will create a temporary variable that will be automatically deleted on the next Wimp_Poll loop.

TWimp_MiscOp, "Del-vars"

Entry:	R0	Pointer to string "add-var", "set-var"
	R1	Pointer to name of variable
	R2	Pointer to string of variable value

This SWI will delete all temporary variables currently stored. Permanent variables will not be deleted.

TWimp_MiscOp, "Del-vars-perm"

Entry: R0 Pointer to string "add-var", "set-var"
 R1 Pointer to name of variable
 R2 Pointer to string of variable value

This SWI will delete all variables currently stored. All permanent and temporary variables are deleted.

Using Variables in Your Program

To use a variable, surround it with the { and } characters with no spaces. They are case sensitive.

```
SYS "TWimp_MiscOp","add-var","window","the_window_name"  
SYS "TWimp_Create","<window {window}> size:400,400"  
SYS "TWimp_Open","{window}"
```

Variables can be used in many places. They can be used with and even contain register values too.

```
SYS "TWimp_MiscOp","add-var","window","the_w_name"  
SYS "TWimp_MiscOp","add-var","icon","the_i_name"  
  
SYS "TWimp_Update","{window}{icon}","text:'%2'",text$
```

You should not use other variable names inside a variable definition. This is undefined and can lead to crashes in the current version of TWimp.

Note that you can use variables when loading files. The variables must be set before loading.

Any variable that is not found will not be replaced and you will be left with the variable name.

Using Variables in Actions

Variables can be set in your actions.

```
<button> text:"Open Window"  
          on-click:-set-var wind_name, mainwindow  
          on-click:OPEN
```

There are two on-click actions and both will be run. The TWimp ones will be run first, then your OPEN will be returned to your program.

```
CASE reason$ OF  
  WHEN "OPEN" THEN SYS "TWimp_Open","{wind_name}"  
ENDCASE
```

Reading Variables Back into Your Program

To read the value of a variable, use MiscOp with the read-var parameter.

TWimp_MiscOp, “Read-var”

Entry:	R0	Pointer to string “read-var”
	R1	Pointer to name of variable
Exit:	R0	Pointer to string of the variable value

TWimp_MiscOp, “Read-var-value”

Entry:	R0	Pointer to string “read-var”
	R1	Pointer to name of variable
Exit:	R0	Numeric value of variable converted from the string

Global Settings

Global settings are set in a <global> definition.

On-load actions can be set in the global definition and this is described in the chapter regarding saving and loading file.

Language Support

Text definitions that are for icons, windows, menus, menu items and more can take on language definitions, including for sprites.

Each item has a tag that refers to the translated item.

```
// Define an icon
<icon> text:"This is the default text"/LOOKUP
```

The above definition defines the default text as "This is the default text". The default language does not have to be English. After the text definition is a forward slash and the following part is called a 'tag'.

Swapping the Language

Dictionaries can be created via TWimp to swap the default text with a definition in the dictionary with the tag LOOKUP.

This is still a work in progress and dictionaries are due to change so the format is not currently listed here.

Using Tags

Tags are stored against your icons and other items. This means you can read which tag was used when the text was last updated. Note that tags can be lost if your icon is writable and the user changes the text.

```
// Create a pop-up icon with a menu
<popup month> pop-up-list:    "Months"/MONTHS ,
                               "January"/JAN ,
                               "February"/FEB ,
                               "March"/MAR
```

The above will create a pop-up menu icon with an attached menu. If the users language has a dictionary in the definition files then the months will be translated if the tags exist.

The means reading the icon text in could be in any language.

To keep your code simple, you can read the tag instead which will not change.

```
SYS "TWimp_Read","month","tag" TO tag$
```

The variable tag\$ will now contain "JAN", "FEB" or "MAR" and not the actual text in the icon. By planning forward and using tags instead of reading text you can use this to make multi-language programs much simpler to maintain.

Updating an item by using a tag can be done with the Update SWI.

REM Update an icon, translating if needed

```
SYS "TWimp_Update","theicon","text:'Default'/DEFAULT"
```

This will look up DEFAULT in the dictionary and update the text to the dictionary entry if it exists or using 'Default' if it does not.

The tag for the item will also be updated. Reading the tag of that icon will now return the string DEFAULT no matter what the actual text in the icon is.

If you do not provide the tag when updating an item then the tag will be discarded since TWimp will assume this new text is now something different to the dictionary entry.

Although you can update using just the tag it is not recommended since the text will update to be nothing if there is no dictionary entry. Unless this is what you require, it should be avoided.

REM This should be avoided

```
SYS "TWimp_Update","icon","text:/MESSAGE"
```

This will lookup MESSAGE in the dictionary. If it is not found it will use the provided string (there was none) and will take the text out from your icon.

This is reserved for future use and it may be that it will lookup users secondary languages or find closely linked languages so please do use this format at the moment. As an example, it may be that US English will lookup up UK English if the definition does not exist so that the US translations may only need to be done for the few spelling differences and not every single translation would be needed. The same for French and Canadian French may happen. It may replace the text with a zero-length string today but it may not function like that in the future.

English does not have to be the language you work with in the original definition files if you prefer or English is not your primary marker or first language. You can have English added as a translated language at a later date.

Scripts

TWimp can handle scripting. This is done using the <script> block with a name. It can then be called by your program or within a TWimp action. Commands are the same as standard TWimp actions

```
// Make a script called openMe
// This opens a window and greys an item called OpenButton
<script openMe>
    -open mainwindow
    -update OpenButton "grey"

// Make a close version that does the opposite
<script closeMe>
    -close mainwindow
    -update OpenButton "no-grey"
```

To call the scripts, you can use standard TWimp actions in your definitions or by calling them from your program.

```
// From an icon definition
<button OpenButton>
    text:"Open Window"
    on-click:-script openMe

REM From your program
SYS "TWimp_Action","openMe"
```

TWimp_Action (Run Action)

Entry: R0 Action name
 R1 Item name or zero

The item name will be passed to the script and can be used in -update where the item name has not been defined in the script.

TWimp_Action (Simulation)

Entry: R0 Item name
 R1 Action type
Exit: R0 Reason string or empty string if none

This will run the action of an icon, window, menu or menu item that has already been defined. This can help reduce duplicate code by running a script assigned to an item. The reason string will be returned in R0 and not your Poll block. Action type can be any of click, select, menu, adjust, hover, double-click, double-adjust, drag, drag-adjust, load, change, open, close, enter, leave, gain-caret, lose-caret.

Copying Designs

Icons, windows, menus and more can have the design copied from one to a newly created one of the same type of icon. This can be done by adding '-copy' after the name when creating an item followed by the name of the item you wish to copy. It must be of the same main type to be able to be copied. Any icon type can be copied into any other icon type but you cannot copy an icon type into a window type.

Most design elements of the item will be copied but some things like the actions will not be copied and you must add those manually for each.

```
// Make an icon with a red background, yellow text, 3D border
<icon thename> text:"Hello"
                colour:yellow bg-colour:red
                border:out
                on-click:CLICKED

// Create a second icon with a red background, yellow text
// and 3D border
<icon secondicon -copy theicon> text:"Goodbye"
```

The above definitions will create an icon called 'thename' with the text Hello. The colours are defined and the on-click event is defined too.

The next definition will create another icon called 'secondicon' which copies the design of 'theicon'. This copies the text, the colours, borders and a lot more. It does not copy the on-click event. You may note that this copies the text, which contained the word Hello. The definition then overrides that copied part, replacing Hello with the word Goodbye for the second icon. That allows you to copy and then modify the copied icon.

Note that icon you are copying must have already have been defined. You cannot swap the above two definitions around. They can be copied from one window to another and the same applies to menu items.

Icon Positioning to a Window

An icon position can be defined to fit the extent of the window. These can be set at startup or moved automatically if you adjust the size of a window. You can also set an icon to be positioned in the visible area so the icon will usually be visible to the user.

Position an Icon to a Window Corner

To position an icon to a window corner, specify the position, width and height to the corner required.

```
// This will be at the bottom-right of the window
<ok> bottom-right:20,20
```

The position parameters are the number of OS units in from the bottom right of the window. You can optionally add a height and width to the parameter list if you want to override any default sizes.

Top-left, top-right, bottom-left and bottom-right are all provided.

Automatically Updating if the Window Size Changes

The four corner parameters can have be automatically updated by TWimp if you change the extent of the window.

```
// Auto has been added to the end for automatic updating
<ok> bottom-right-auto:20,20
```

Now if you change the size of the window this icon is in, the icon will automatically be moved too.

```
REM Change the size of the window, no need to change icon
REM position - this will be done automatically
SYS "TWimp_Update","mainwindow","add-height:400"
```

Keep an Icon in the Visible Area

TWimp can keep an icon in the visible area instead of the work extent of a window. This is the part of the window that the user sees on the screen.

```
// Keep in the visible area
<ok> bottom-right-visible:20,20
```

If the user moves or resizes the window, this icon will be automatically moved into the position requested.

Advanced Icon Positioning

Icon positions can be placed relative to other icons. This can be done by referring to a specific icon name or by using the position of the last created icon.

A base value for the window can also be set. This is used to offset all icons made after this is changed by a specific amount.

Frame names can be used to surround a set of icons and you can refer to the name of the frame to set the position of your new icon. The icons in the frame must have already been defined. Group names can also be used.

Last Icon Positioning

A number of values can be used in the position type keywords. These include lastx, lasty, lastleft, lasttop, lastbottom, lastright, lastwidth and lastheight. Each of these can have one simple addition or subtraction value following these keywords.

```
// Create an icon with a position
// This has an x position of 50, y of 150
// width of 300 and height of the default value for the type
<display icon1> text:"Hello" pos:50,150,300

// Create a second icon with a position under the last
<display icon2> text:"Hello"
                pos:lastx,lastbottom+12,lastwidth,lastheight

// And a third. The position line is identical but will make
// this line under icon2
<display icon3> text:"Hello"
                pos:lastx,lastbottom+12,lastwidth,lastheight
```

The position is set to the last x value for the new x value and the new y value is at the bottom of the last icon made plus another 12 units. You can mix and match using standard numbers and these special values.

Note that these last values do not take into account the pre and post icons and you should use frame icon positioning if you wish to define a position taking this into account.

Icon Relative Positioning to Specific Icon

To use a value of an existing icon, you can use the following where you should replace the ... with an icon name:

- icon[...].x

- icon[...]y
- icon[...]left
- icon[...]right
- icon[...]top
- icon[...]bottom
- icon[...]width
- icon[...]height

All of these can have additions or subtractions following.

```
// Align to 12 units to the right of an icon called 'volume'
// with a height of 52 and a width of 100
<display> text:"Volume"
          pos:icon[volume]right+12, icon[volume]y ,100,52
```

Note that these values are the core icon coordinates and do not include the pre and post labels, if they are defined on your icons. To take these into account, you can use the bounding[...]x format instead of icon[...]x.

Frame Icon Positioning

Frame icon positioning is the same as the icon relative positioning above, except that you use frame[...]x instead of icon[...]x.

The difference is that a frame will calculate the x, y and other positions and sizes from all icons with that frame name and takes pre and post into account.

The ... should be replaced by a frame name.

All icons can have a frame name. This can be set by setting the icon frame in the definition or by using set-frame and unset-frame around a number of icons.

```
// Start a frame called FR
set-frame:FR
<display> text:"Icon 1" pos:50,50,300,52
<display> text:"Icon 2" pos:150,150,400,52

//Stop following icons from having the frame name set
unset-frame:FR

// Make an icon to the right of the frame. This takes
// the position of all icons with this frame name into
// account when positioning. So frame[FR]right will be
// a value of the right of all icons in the frame
// whichever is the most on the right-hand side
<display> text:"Icon 3"
          pos:frame[FR]right+12,frame[FR]y
```

Group Icon Positioning

This is identical to frame positioning except it applies to icons with group icons, typically radio buttons. Simply use `group[...]x` and any other similar format.

Pre and Post Icon Positioning

You can refer to the positions of pre and post icons using `pre[...]x` and `post[...]x` and similar with a specific icon name.

Bounding Icon Positioning

Using `bounding[...]x` and similar with a specific icon name will return the values in the same way that `icon[...]x` does, except that pre and post icon are taken into account. This means that `bounding[...]left` and `bounding[...]right` will give you the full bounding box of that icon and `bounding[...]width` will give you the full width of the icon including both pre and post icons if they exist. Some pre and post icons may appear in other places such as top and bottom and bounding will take this into account.

Error Messages

Error messages are much easier to make under TWimp than the traditional Wimp. The SWI TWimp_Error is provided and takes just two parameters. The formatting string can be omitted by sending a null string or a zero value.

TWimp_Error

Entry:	R0	Pointer to error string
	R1	Pointer to error formatting string or zero
Exit:	R0	String value of button pressed if in a TWimp task
	R1	String value of tag pressed if in a TWimp task
	R2	Number value of button pressed

The error formatting string can contain the following:

- ok
- cancel
- cancel:default
- extra:<extraButtonsTextCommaSeparated>
- sprite:<spriteName>
- info, warn, stop or question
- beep or no-beep

The following creates a nice error message.

```
SYS "TWimp_Error","The error message",  
    "ok cancel:default sprite:file_fff stop"
```

This has both the ok and and cancel buttons but the cancel button is the default option if the user presses enter. There is a sprite and the standard error icon is replaced with a stop icon instead.

The return string in R0 will be one of the following, without quotes:

- "OK" if ok or continue was selected
- "CANCEL" if cancel was selected
- The string from the extra button which was selected
- A null string if nothing was selected

Extra buttons can be created on many versions of the Wimp. These can be supplied to TWimp in a list of strings that are comma separated and may

include language tags. The language tag string will be returned as a pointer in R1.

```
SYS "TWimp_Error","An Error",  
    "extra:'Button 1'/B1TAG,'Button 2'/B2TAG"
```

This SWI can be called from outside a task. If you do this, R0, R1 and R2 will return the standard Wimp numeric value of the button pressed and not the string values. This can be useful for creating a standard error box with a single button, especially in an error handler where you may be outside the declared TWimp task start and end.

Claiming Memory

TWimp has a facility to allow you to claim memory for your application during runtime. This is claimed in a dynamic area where possible or the module area if a dynamic area is not available.

The advantage of using TWimp to claim memory this way is that all this memory should be freed when your task ends, including if your task crashes.

There are no limits on the number of memory claims you make or the size of memory you require.

The V flag will be set if a memory allocation fails. Your program must handle this.

SWI TWimp_MiscOp,"Claim"

Entry:	R0	Pointer to string "claim"
	R1	Size of memory required in bytes
Exit:	R0	Memory pointer of claimed area
	V	The V flag will be set if claim failed

SWI TWimp_MiscOp,"Release"

Entry:	R0	Pointer to string 'release'
	R1	Memory pointer of area that was claimed

Please note that TWimp does not check if you are releasing a piece of memory twice and you may cause a crash if you attempt this.

REM Recommended claim and release code

```
mem% = 0
```

REM Claiming code, handling a memory claim error

```
SYS "TWimp_MiscOp","claim",1024 TO mem%  
IF mem%=0 THEN PROChandlememoryerror
```

REM Release code, resetting mem% so can't release again

```
IF mem%<>0 THEN SYS "TWimp_MiscOp","release",mem%  
mem% = 0
```


Programming in C, Using FPE and Poll Words

Programming in C

All of the TWimp SWIs that return strings will return an empty string if the item is not found or available.

In some languages you may prefer this to return a NULL string pointer instead. This can be done by calling TWimp_ReturnZero. Once called your task will get all non-available text strings returned as zero (or NULL).

TWimp_ReturnZero

Takes no parameters.

Using the FPE

By default TWimp will not save the FPE (Floating Point Emulator) registers for your task. If you do require this call MiscOp with FPE as the reason string.

TWimp_MiscOp,"FPE"

Entry: R0 Pointer to string "FPE"

This will inform TWimp to save the registers for the floating-point for your task when switching. Most tasks will not require this.

Poll Words

To set a non-zero module poll word in your task called MiscOp with "Poll-Word" as the reason string.

TWimp_MiscOp,"Poll-Word"

Entry: R0 Pointer to string "poll-word"
 R1 Pointer to the memory address of the poll-word

When the pollword is non-zero your reason string returned will be "_POLLWORD" in your program and you should respond to that. Please note the underscore at the front of the string.

Debugging and Help

TWimp will allow you to debug names of icons by using the help system. Load the Help application or a similar program and use the debug-names option in the MiscOp SWI.

TWimp_MiscOp, “Debug-names”

Entry: R0 Pointer to text “debug-names”

Once this has been called in your program, the help messages will give some information about your windows, icons, menus and menu items including their names.

Sixteen Colour Modes

To view what your application would look like using just the 16 basic Wimp colours, set the variable TWimp\$16Col to 1 and restart your application. This option must not be set in any distributed software.

General Information

Reading the Screen Size

To reason the screen size use MiscOp,"screen-size"

TWimp_MiscOp,"Screen-size"

Entry:	R0	Pointer to string "screen-size"
Exit:	R0	X size of screen
	R1	Y size of screen

My Task Handle

To remind yourself of your own task handle use MiscOp,"task-handle"

TWimp_MiscOp,"Task-handle"

Entry:	R0	Pointer to string "task-handle"
Exit:	R0	Your task handle

Renaming Items

This can rename an iconbar, window, menu, icon or menu item. It will return an error if the item name does not exist.

TWimp_MiscOp,"Rename"

Entry:	R0	Pointer to string "rename"
	R1	Pointer to string of existing item
	R2	Pointer to new name of item

Swapping the Names of Items

You may wish to swap the names of two items.

TWimp_MiscOp,"Swap-name"

Entry:	R0	Pointer to string "swap-name"
	R1	Pointer to string of existing item
	R2	Pointer to string of another existing item

Working with Risc OS 2 and Risc OS 3.11

A limited amount of testing has been provided for older versions of Risc OS back to 3.71. There is no guarantee it will run successfully on older versions of the operating system.

The first issue with older versions of Risc OS is the fact that Wimp_PollIdle may not exist. TWimp can help let you know if you should call Poll or PollIdle by returning the SWI number for the Wimp call that is recommended.

TWimp_MiscOp,"Poll-SWI"

Entry: R0 Pointer to string "poll-swi"
Exit: R0 SWI number of Wimp_Poll or Wimp_PollIdle

A further issue with Risc OS 2 is the task handle must be sent into TWimp on every poll loop.

TWimp_MiscOp,"Set-task"

Entry: R0 Pointer to string "set-task"
R1 Task handle

Your task handle is returned in R0 when you call TWimp_Start or TWimp_StartOnce. You cannot use TWimp_MiscOp to read the task handle in Risc OS 2.

A typical Risc OS 2 and Risc OS 3.11 startup and poll loop would look like the following code.

```
SYS "TWimp_Start","My Application" TO taskhandle%
SYS "TWimp_MiscOp","poll-swi" TO swi%

REM load in files, startup procedures here
REPEAT
    SYS "TWimp_PollSetup" TO a%,b%,c%
    SYS swi%,a%,b%,c% TO a%,b%,c%
    SYS "TWimp_MiscOp","set-task",taskhandle%
    SYS "TWimp_PollAction",a%,b%,c% TO reason$
UNTIL reason$="QUIT"
SYS "TWimp_End"
END
```

The code should allow the task to run on older versions of the OS and will not cause any issues with newer versions.

Some TWimp functionality and Wimp functionality is not available in older OS versions. The nested Wimp is currently used for toolbars and will require the user to soft-load a newer window manager on all versions of Risc OS 3 lower than 3.50 and the will not work at all on Risc OS 2.

TWimp will still be able to move icons and change data but you should expect slow performance on older machines due to it having a higher chance of having to rebuild entire windows and redrawing the window on top of the slower processor.

Although TWimp should function on most versions of Risc OS it is not supported. You can report where things do not work on anything lower than Risc OS 3.50 but there is no guarantee that they will or can be fixed.

TWimp is a large module and you may encounter issues with loading the module unless you have a machine with a reasonable amount of memory or are emulating with a good memory size.

SWI Numbers

TWimp can be provided in two formats. There is the demo version TWimpDemo and the full version TWimp. These have their own SWI numbers.

This lists both versions of the SWI call with the first part as the top header and the second part down the side and the values are in hexadecimal. TWimp_Start is hexadecimal 5A700.

	TWimp	TWimpDemo
Start	5A700	5A740
StartOnce	5A701	5A741
End	5A702	5A742
PollSetup	5A703	5A743
PollAction	5A704	5A744
Load	5A705	5A745
Create	5A706	5A746
Open	5A707	5A747
OpenAt	5A708	5A748
OpenAsMenu	5A709	5A749
Close	5A70A	5A74A
Update	5A70B	5A74B
UpdateIf	5A70C	5A74C
UpdateText	5A70D	5A74D
UpdateValue	5A70E	5A74E
UpdateSprite	5A70F	5A75F
UpdateNumber	5A710	5A750
Read	5A711	5A751
ReadText	5A712	5A752
ReadValue	5A713	5A753
ReadSprite	5A714	5A754
ReadGroup	5A715	5A755
PollData	5A716	5A756
ReturnZero	5A717	5A757
Loaded	5A718	5A758
Saved	5A719	5A759

	TWimp	TWimpDemo
Error	5A71A	5A75A
AddTimer	5A71B	5A75B
RemoveTimer	5A71C	5A75C
SaveOptions	5A71D	5A75D
LoadOptions	5A71E	5A75E
Delete	5A71F	5A75F
Event	5A720	5A760
Action	5A721	5A761
MiscOp	5A722	5A762
Enumerate	5A723	5A763