northgate
INFORMATION SOLUTIONS

UIMS

Version 2.0

DATA/BASIC API Reference Manual

Document No. UM70006277B
September 1994

Northgate Information Solutions UK Limited
Boundary Way
Hemel Hempstead, Hertfordshire,
HP2 7HU

Tel: +44 (0)1442 232424
Fax: +44 (0)1442 256454

www.northgate-is.com

Comment Sheet

Please give page number and description for any errors found:

Page	Error

Please use the box below to: describe any material you think is missing; describe any material which is hard to understand; enter any suggestions for improvement; provide any specific examples of how you use your system which you think might be useful to readers of this manual

Continue on a separate sheet if necessary.

If you would like someone to contact you about documentation, please tick this box. ☐

Important: *Please enter your name, address and telephone number on the back of this form before returning.*

Manual: UIMS DATA/BASIC API, Reference Manual, UM70006277B, September 1994.

Name of Sender: ______________________________

Company Name / Address: ______________________

____

____

Telephone: ___________________________________

Date: ____

FOLD B

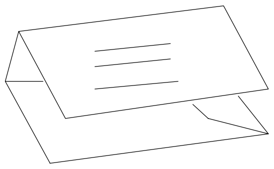
BUSINESS REPLY SERVICE
Licence No. HH698

2

F
O
L
D

A

Technical Publications Department
Northgate Information Solutions UK Limited
Boundary Way
Hemel Hempstead
Hertfordshire HP2 7HU



Fold along fold A, then B and C. Tuck in so name and address are on outside.

FOLD C

Appendix B Screen Colours

Appendix C Resource Compiler Keywords

Appendix D Error Codes

Glossary

Index

List of Figures

List of Tables

Chapter 1
About this Manual

This chapter describes the different sections of this manual and any conventions used.

Purpose of this Manual

This manual is intended for the DATA/BASIC programmer who will be writing applications that use the REALITY User Interface Management System (UIMS). It gives both general and detailed information about the UIMS subroutines, messages and resource compiler. It does not attempt to explain how to create a UIMS application. Rather, it gives detailed descriptions of each component of the UIMS DATA/BASIC API for readers who already have a basic understanding of DATA/BASIC programming.

It is assumed that, in addition to being an experienced DATA/BASIC programmer, you will be familiar with RealLink for Windows and Microsoft Windows, and have access to the appropriate user manuals.

This manual consists of the following sections:

Chapter 1, About this Manual, describes the different sections of the manual and any conventions used.

Chapter 2, Overview, gives a brief overview of UIMS.

Chapter 3, Objects, describes the UIMS objects and indicates which subroutines can be used to manipulate each of them.

Chapter 4, Messages, describes how a UIMS application uses messages to receive user input. It also lists the different types of message and gives details of their parameters.

Chapter 5, NewView, describes the UIMS NewView subsystem for enhancing existing applications.

Chapter 6, Subroutine Reference, lists the UIMS subroutines in alphabetical order. It gives the full syntax for each subroutine and provides details of parameters and return values.

Chapter 7, Resource Compiler, describes how to use the UIMS Resource Compiler to create resource files on the PC.

Chapter 8, Help System, describes how to provide the user of a UIMS application with on-line help.

Appendix A, Key Aliases, lists the symbolic constant names, decimal values and descriptive information for the UIMS key aliases.

Appendix B, Screen Colours, describes how screen colours are specified in a UIMS application and lists the pre-defined logical colours. It also explains the effects of the different graphics drawing modes.

Appendix C, Resource Compiler Keywords, lists the object type and attribute keywords recognised by the resource compiler and gives details of mandatory attributes and valid attribute settings. It also lists the error messages that might be displayed by the resource compiler and suggests probable causes for these.

Appendix D, Error Codes, lists the completion and error codes which might be returned by UIMS subroutines.

Related Documents

UIMS DATA/BASIC API, Quick Reference Guide.

UIMS DATA/BASIC API, Programmer's Guide.

RealLink for Windows User Manual.

REALITY DATA/BASIC Reference Manual.

Microsoft Windows User's Guide.

Conventions

The following conventions are used in this manual:

`Text`	Bold text shown in this typeface is used to indicate input which must be typed on the keyboard.
`Text`	Text shown in this typeface is used to show text that is output to the screen.
Bold text	Bold text in syntax descriptions represents characters typed exactly as shown. For example: **WHO**
Text	Characters or words in italics indicate parameters which must be supplied by the user. For example, in **GetChildFocus**(*Context*, *Contact*, *vChild*) the parameters *Context*, *Contact* and *vChild* are italicised to indicate that this is the general form of the **GetChildFocus** subroutine. In an actual program, the user supplies particular arguments for the place-holders *Context*, *Contact* and *vChild*. Italic text is also used for titles of documents referred to by this document.
vText	A lower case '*v*' prefixing a place-holder name indicates that a variable must be supplied so that a value can be returned. In the example above, for instance, the '*v*' prefix to the parameter name *vChild* indicates that, in an actual program, the user must supply the name of a variable in which to return the handle of the child which currently has the focus.
aText, *vaText*	A lower case '*a*' prefixing a place-holder name indicates that either the programmer must supply a dynamic array or, when combined with a lower case '*v*', that on return the parameter will contain a dynamic array with one value in each attribute.
[Brackets]	Brackets enclose optional parameters. For example, in

#IFDEF *ident*
    source code block
[**#ELSE**
    source code block]
#ENDIF

the keyword **#ELSE** and an associated *source code block* can optionally be included.

…	In syntax descriptions, ellipses following a group of items indicate that the parameters preceding can be repeated as many times as necessary. For example, in *ATTRIBUTE* = *Value* [*ATTRIBUTE* = *Value* …] the ellipses indicate that the sequence *ATTRIBUTE* = *Value* may be repeated as many times as necessary. Vertical ellipses are used in program examples to indicate that a portion of the program is omitted.
SMALL CAPITALS	Small capital letters are used for the names of keys such as RETURN.
CTRL+X	Two (or more) key names joined by a plus sign (+) indicate a combination of keys, where the first key(s) must be held down while the second (or last) is pressed. For example, CTRL+X indicates that the CTRL key must be held down while the X key is pressed.
Enter	To enter means to type text then press RETURN. For instance, 'Enter the WHO command' means type **`WHO`**, then press RETURN. In general, the RETURN key (shown as ENTER or ↵ on some keyboards) must be used to complete all terminal input unless otherwise specified.
Press	Press single key or key combination but do not press RETURN afterwards.
X'*nn*'	This denotes a hexadecimal value.

User Comments

A Comment Sheet is included at the front of this manual. If you find any errors or have any suggestions for improvements in the manual please complete and return the form. If it has already been used then send your comments to the Technical Publications Manager at the address on the title page.

Chapter 2
Overview

This chapter gives a brief overview of the REALITY User Interface Management System (UIMS). It describes the main features of UIMS, the UIMS software, the three types of UIMS application, and the objects and contacts that make up a UIMS graphical user interface.

Introduction

RealLink for Windows is a PC terminal emulator that runs in the Microsoft Windows environment. At the heart of RealLink is a User Interface Manager that provides its interface to the Windows environment and generates its graphical display.

RealLink makes many of the features of the User Interface Manager available to host applications by means of commands that can be transmitted across a LAN or other communications link. The UIMS DATA/BASIC API provides the REALITY DATA/BASIC programmer with a suite of subroutines that can be used in applications. These subroutines simplify the programmer's task by constructing the User Interface Manager commands and transmitting them to RealLink. RealLink, in turn, carries out these commands and returns any results to the host application via variables supplied by the DATA/BASIC programmer.

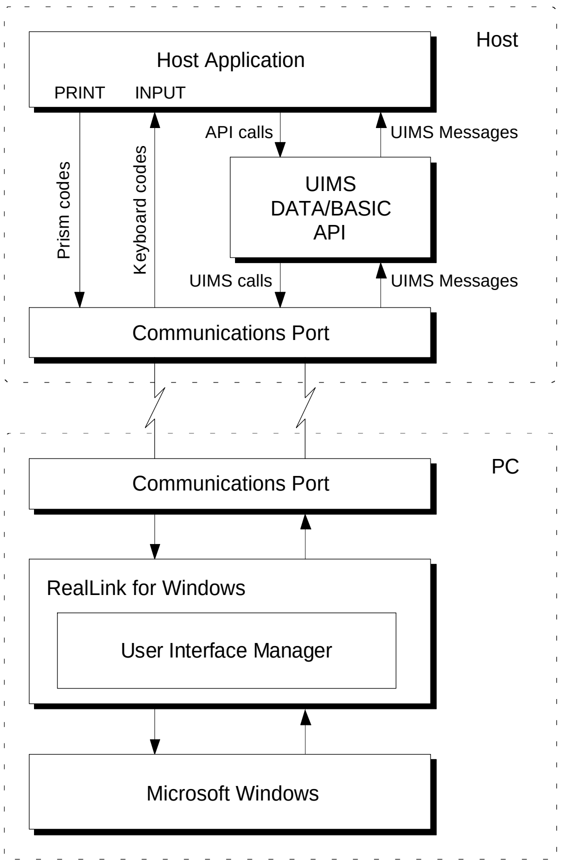


Figure 2-1. The User Interface Management System

By using the UIMS DATA/BASIC API, programmers can create applications on MDIS Series 19 and Series X host systems which make use of the features provided by the Microsoft Windows graphical user interface. These include:

- A graphical user interface featuring windows, menus, dialog boxes and controls for applications.
- Queued input.
- Multitasking.
- Data interchange between applications.

UIMS software

The UIMS software consists of the following components:

- RealLink for Windows.
- An Application Programming Interface for DATA/BASIC (DATA/BASIC API). This consists of a suite of cataloged DATA/BASIC subroutines which provide the commands that host applications use to access the RealLink User Interface Manager.
- A resource compiler for use by application programmers. This allows the graphical objects used by an application to be defined on the PC rather than the host, thus improving performance by sharing the processing and reducing communication between the two systems. In addition, resources created in this way are loaded only when the application is run, allowing a programmer to produce different versions of an application, without having to change the host program.

When developing UIMS applications, you will require all three of the above. The users of your finished applications, however, need only the first two, but they will require copies of your compiled resource files (on their PCs), in addition to your host programs and subroutines.

UIMS Applications

There are three types of application program which make use of the UIMS DATA/BASIC API: true UIMS applications, 'hybrid' applications, and NewView applications.

- A UIMS application is one which uses only the advanced user-interface functions of the RealLink software for input and output.
- A hybrid application is a character-display application whose presentation has been improved by the addition of some advanced user-interface functionality, but which still relies largely on standard character input and output for its user interface.
- A NewView application is similar to a hybrid application, in that it is a character-display application whose presentation has been improved. However, NewView allows the existing user interface to be converted, so that the changes to the original code are minimised.

The *UIMS DATA/BASIC API, Programmer's Guide* describes how to write these three types of application.

Objects and Contacts

The user interface for a UIMS application is built up of various kinds of pre-defined building block (objects). Each of these objects acts as a template for creating graphical elements which share certain common characteristics; for example, every list box is a box containing a list. Characteristics such as size or colour can be changed to suit the requirements of the user interface.

There are two types of object: contacts (windows, buttons, list boxes, etc.) which can be displayed on the screen, and which provide different types of interface with the user; and objects which define the appearance of the contacts (screen colours, text font and style, line width, etc.). Chapter 3 describes each of the UIMS objects and contacts in detail.

The programmer designs the user interface for an application by creating contacts of the required types which are then displayed on the screen as appropriate to the requirements of the application. For instance, when the application requires input from the user, it might display a dialog box, which could contain an input field and option buttons to allow the selection of various options. Command buttons would allow the user to accept any changes or to cancel the operation.

Graphics Contacts

UIMS provides subroutines to draw text, lines and rectangles. If these are used, however, the host application must ensure that they are redrawn when necessary (for instance, when the user switches from one application to another, thus exposing all or part of a window). Since the necessary commands must all be sent from the host to the PC via the communications link, this can result in a slow response.

The alternative is to use the UIMS graphics contacts. These provide an interface to the user only in that they can be displayed on the screen. However, they are always redrawn automatically by UIMS when ever necessary. This reduces the communication between the host and the PC, and improves the speed of your application.

Contact Hierarchy

When you use UIMS contacts, you must organise them in a hierarchy consisting of 'parents' and 'children'. This hierarchy has the following rules:

- A contact can have only one parent. Attaching a contact to a new parent removes it from its previous parent, if any.
- A child contact can, itself, be the parent of other contacts.
- A contact cannot be displayed on the screen unless it has a parent contact. Similarly, the children of a contact that has no parent cannot be displayed.

- A contact cannot be displayed unless its parent is visible.
- A child contact can only be displayed in the area of the screen that is occupied by its parent contact. If it is positioned so that it overlaps the edge of its parent, only the part that is inside the parent will be displayed.
- A child contact is always positioned relative to its parent. If the parent moves, its children move with it.
- Disabling a contact also disables its children.
- Destroying a contact also destroys its children.

Chapter 3
Objects

UIMS provides various kinds of graphic objects with which to create the user interface for an application. This chapter describes these objects and indicates which subroutines can be used to manipulate each of them.

Common Contact Attributes

There are a number of attributes which are common to almost all contacts. The following lists these attributes and the subroutines that control them. Note that where an attribute is not supported by a particular contact, this is mentioned in the contact description.

Size	The overall width and height of the contact. Subroutines – **Resize**, **GetSize**.
Position	The position of the top left-hand corner of the contact, relative to the top left-hand corner of its parent. Subroutines – **Move**, **GetPosition**.
Border style	Whether or not a window has a visible border. In the case of an App window, the type of border (single or double) is determined by the style of the window. Subroutines – **SetBorderStyle**, **GetBorderStyle**.
Help index	The name of the help file section with which the contact is associated (see page 3-6). Subroutines – **SetHelpIndex**, **GetHelpIndex**.
Enabled	Whether or not the contact is enabled. A disabled contact is displayed on the screen, but cannot be selected by the user. The disabled state is indicated by a greying effect, the exact form of which is platform dependent. Subroutines – **SetEnabled**, **Disable**, **Enable**, **GetState**.
Visible	Whether or not the contact is visible on the screen. Subroutines – **SetMapped**, **Map**, **UnMap**, **GetState**.

Update mode	Specifies when a contact will be redrawn if a change occurs. The following options can be selected. • Immediate – redraw immediately. • None – don't redraw; wait for a **Draw** command. The **Draw** subroutine redraws the specified contact immediately, whatever its update mode setting. Subroutines – **SetUpdate**, **GetUpdate**, **Draw**.
Event mask	A list of message types that will be passed on by the contact to its parent (refer to Chapter 4 for details). Subroutines – **SetEventMask**, **GetEventMask**.

AppContext

One of the first subroutine calls in a UIMS application must be to **SignOn**. This starts a UIMS session and creates an **AppContext** object containing various configuration settings.

Once created, the App context is unique to the instance of the application that created it. The user could run a second instance of the same application, but this would have its own App context which might be configured differently.

All **AppWindow** contacts created by the application must be children of the **AppContext**.

Attributes

Root window	The handle of the first **AppWindow** contact created by the application. Subroutines – **GetRootWindow**.
Front window	The handle of the **AppWindow** which either currently has the focus or which contains the contact which currently has the focus. If some other application has the focus, the front window is that which last had the focus. Subroutines – **GetFrontWindow**.
Coordinate mode	The coordinate system used to specify the positions and sizes of contacts. Two modes are available: text (character) or graphics (pixel). In text mode, a character cell is the size of an average character in the default (system) font. Subroutines – **SetCoordMode**, **GetCoordMode**.
Drawrule	The handle of the default **Drawrule** object. This has default **Pen**, **Brush** and **Font** objects as its children. Subroutines – **SetDrawrule**, **GetDrawrule**.
Event mask	A list of message types that will be passed on to the application by the **AppContext** (refer to Chapter 4 for details). Subroutines – **SetEventMask**, **GetEventMask**.

Wait pointer	This provides a simple method of indicating to the user that a lengthy operation is in progress, by changing the mouse pointer to an hourglass (or other wait-pointer, as determined by the hardware platform).
	Subroutines – **WaitPointerOn**, **WaitPointerOff**.
Help file	The name of the current application help file.
	Subroutines – **SetHelpFile**, **GetHelpFile**.
Help key	The key that will be used to display the help text.
	Subroutines – **SetHelpKey**, **GetHelpKey**.

AppHelp

An **AppHelp** object is a compiled Help text file (see Chapter 8) that contains named sections of help text.

The sections of the help file are linked by means of 'hot words' embedded in the text. These act as links to other sections of the file. If the user clicks on a hot word, the associated section of the help file is displayed. The help file also contains an index (built during the compilation process); this contains hot words giving access to every section of the file.

The application can display a specified section of the Help file by calling the **AppHelp** subroutine. The programmer must provide the user with access to the help file; this can be done by creating a Help menu, for example.

The **AppHelp** object also supports context sensitive help. Contacts used within the application can each be linked to a section of the help file. The appropriate section of the help file is displayed whenever the user presses a Help Accelerator key; this is normally function key F1, but can be changed by the application. If the contact is not linked to a help file section, the help index will be displayed.

Subroutines

SetHelpFile	Attaches a help file to the application.
GetHelpFile	Returns the name of the application's help file.
AppHelp	Displays a specified section of the help file.
SetHelpIndex	Associates a contact with a section of the help file.
GetHelpIndex	Returns the name of the help file section which is associated with a specified contact.
SetHelpKey	Assigns a key as the help accelerator.
GetHelpKey	Returns the key currently assigned as the help accelerator.

AppResource

An **AppResource** object is a compiled UIMS Resource Script file (see Chapter 7) that defines a group of UIMS objects and/or contacts. The application can dynamically create all the defined objects and contacts by a single call to the **LoadAppRes** subroutine. An application may load any number of **AppResource** files.

The only attribute of an **AppResource** is its filename.

AppWindow

An **AppWindow** contact is an application's primary interface with the user. Every application must have at least one App window – the Root window; the handle of the root window is always available in the **AppContext** object. An App window must be a child of the App context and it can therefore be displayed anywhere on the screen; it cannot be constrained within the client area of any other window (cf. Child window).

An App window consists of a client area, which must be managed by the application, and a border, managed by UIMS. The border can include a title bar, system menu, maximise and minimise icons, a menu bar, and horizontal and vertical scroll-bars.

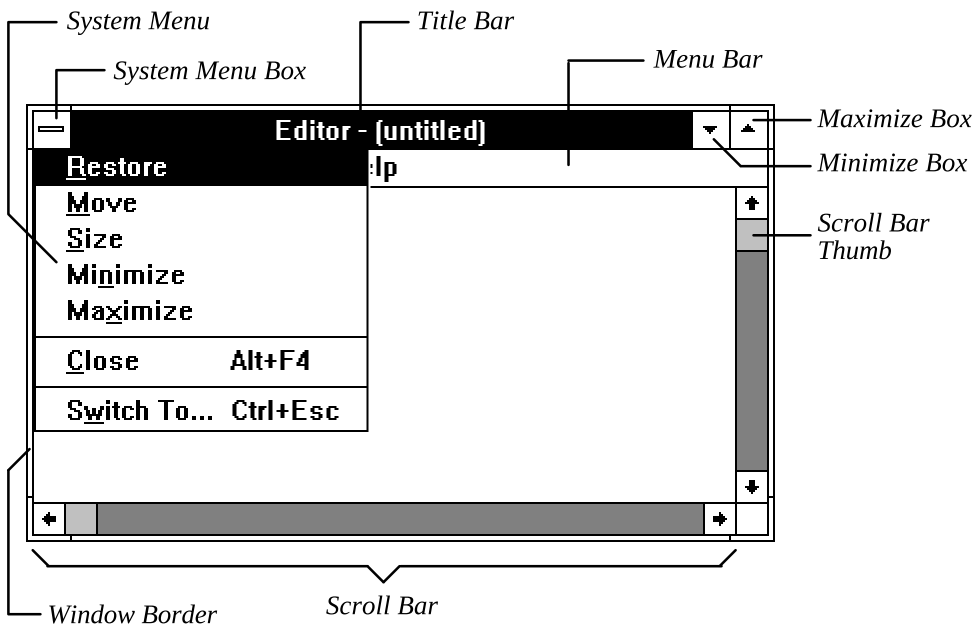


Figure 3-1. The Components of an App Window

An App window is created using the **CreateAppWin** subroutine.

The Client Area

The application has complete control over the appearance of a window's client area. Text, lines and rectangles may be drawn directly on the client area. However, should the client area be disturbed in any way (if, for instance, a dialog box is drawn in the client area) the application must restore it to its previous state.

Text Canvas

If required, the responsibility for maintaining the client area can be partially transferred to UIMS, by specifying that the window should have a text canvas. This is used to hold transient text strings drawn with **DrawTextString** in the window client area, so that they can be repainted at any time. If a window has a text canvas, the application does not have to redraw the text when the window is resized or uncovered by another contact. The default is for a window not to have a text canvas.

Notes:

1. The text canvas only stores the text strings and their positions in the client area; the appearance of the text is determined by the currently selected **Font** object. If the font is changed the appearance of the text will change when it is next redrawn.

2. Graphics shapes drawn with **DrawLine** and **DrawRect** are not stored in the text canvas. The application must ensure that these are redrawn when the window is updated.

3. The **Erase** subroutine can be used to clear the text canvas, and this also clears the whole of the client area. Note, however, that erasing all or part of the client area does not clear the text canvas – the stored text will be redrawn when the window is next updated.

Attributes

Style

The style of the window. This can be a combination of the following options:

- Movable – generates a single border, a title bar, and a system menu with the Move command enabled.
- Closable – this is the same as movable, except that the Close command on the system menu is enabled.
- Iconisable – this is the same as movable, except that the title bar includes a minimise box, and the Minimise command on the system menu is enabled.
- Resizable – this is the same as movable, except that the border is double, the title bar includes a maximise box, and the Size and Maximize commands on the system menu are enabled.
- Display a horizontal scroll-bar.
- Display a vertical scroll-bar.
- Allow movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
- Text canvas (see above).

Subroutines – **AppWinSetStyle**, **AppWinGetStyle**.

Title	The text that will appear in the title bar of the window. Note that an App window only has a title bar if it is movable. If there is no title bar, the title will not be displayed. Subroutines – **AppWinSetTitle**.
Display	The handle of the **Display** object on which the App window will be drawn. This is a screen display object Subroutines – **AppWinGetDisplay**.
MenuBar	The handle of the **MenuBar** contact which is attached to the App window. Subroutines – **AppWinSetMenuBar**, **AppWinGetMenuBar**, **AppWinRemoveMenuBar**.
Horizontal scroll-bar	The handle of the window's horizontal scroll-bar. Subroutines – **AppWinGetHScroll**.
Vertical scroll-bar	The handle of the window's vertical scroll-bar. Subroutines – **AppWinGetVScroll**.
State	Whether or not the window is minimised or maximised. Subroutines – **AppWinSetSizing**, **AppWinMaximize**, **AppWinMinimize**, **AppWinRestore**.
Clip region	Defines a clipping region within the client area for all drawing operations. Text and graphics drawn outside the clipping region are not displayed. Subroutines – **SetClip**, **GetClip**.
Drawrule	The handle of a **Drawrule** object used for all drawing operations within client area. This defines attributes such as foreground and background colours, text font, line width, etc. (see page 3-22). Subroutines – **SetDrawrule**, **GetDrawrule**.

	Pointer	The handle of a **Pointer** object used when the mouse pointer is within the window's client area. Subroutines – **SetPointer**, **GetPointer**.
	Cursor state	The type of cursor displayed in the client area and whether or not it is visible. The following types are available: • Outline cursor (not supported on Microsoft Windows). • Block cursor. • Underline cursor. • Vertical bar cursor. Subroutines – **SetCursorState**, **GetCursorState**.
	Cursor position	The position of the cursor relative to the origin (top left-hand corner) of the client area. The position is specified in text or graphics coordinates, depending on the coordinate mode of the application. Subroutines – **SetCursorPosition**, **GetCursorPosition**.
	Children	The list of child contacts. Subroutines – **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.
	Focus	The handle of the child contact which has the input focus. Subroutines – **SetContactFocus**, **GetChildFocus**.
	Default button	The handle of the default **TitledButton** contact. This attribute is only applicable to windows with the **UIMS.WIN.DIALOG** style. Subroutines – **AppWinSetDefButton**.
Common Contact Attributes	All the common contact attributes (see page 3-2) apply to **AppWindow** contacts.	
Other Subroutines	**DrawTextString**	Draws text on the client area or text canvas.
	DrawLine	Draws a line on the client area.
	DrawRect	Draws a rectangle on the client area.

Scroll	Scrolls the client area.
Erase	Erases a specified part of the client area or the whole of the text canvas.

Brush

A **Brush** object defines the way in which areas of a window client area are filled. A **Brush** cannot be attached directly to a contact, but must be a child of the attached **Drawrule** object.

UIMS provides a default **Brush**, the handle of which can be obtained by using **GetDrawrule** to fetch the handle of the drawrule for the Application context, and then calling the **DrawruleGetBrush** subroutine. Additional **Brush** objects can be created with the **CreateDrawBrush** subroutine.

Attributes

Colour

A UIMS logical colour or RGB value. Note that this attribute specifies only the foreground colour of the brush pattern; in use, the background colour will be determined by the **Drawrule** to which the brush is attached (see Figure 3-2).

Subroutines – **BrushSetColour**, **BrushGetColour**.

Style

The style of the brush. The following styles are available:

UIMS.BRUSH.SOLID
A solid block in the specified foreground colour.
UIMS.BRUSH.HOLLOW
Transparent. The colour attribute is ignored.

Subroutines – **CreateDrawBrush**.

Other Subroutines

DrawruleGetBrush
Returns the handle of the **Brush** which is attached to the specified **Drawrule** object.

DrawruleSetBrush
Attaches a **Brush** to a **Drawrule** object.

CheckButton

A **CheckButton** is a contact that allows the user to select and deselect an option. It consists of a small box with a button title to the right. When the option is selected, the box contains a mark of some kind – usually a cross or a tick, depending on the platform.

A check button differs from an option button in that, when a number of check buttons are grouped together, each button can be selected independently of the others.

A **CheckButton** contact is created with the **CreateCheckButton** subroutine.

Attributes

Title	The text that will appear beside the check button. One of the characters in the title can be designated as a selector key by preceding it with an ampersand character. Subroutines – **CheckButtonSetTitle**.
State	Whether or not the button is selected. Subroutines – **CheckButtonSetSelected**, **CheckButtonSelect**, **CheckButtonDeselect**, **CheckButtonGetSelected**.
Autotoggle	This is an operating mode that removes the burden of check mark control from the application. When selected, Autotoggle automatically toggles the check mark on or off, as appropriate, each time the user selects the button. Subroutines – **CheckButtonSetToggle**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to **CheckButton** contacts.

ChildWindow

A **ChildWindow** contact is similar to an App window, but its movement is constrained within the client area of its parent. Its position is specified relative to its parent, so that, when the position or size of the parent window changes, the Child window will be redrawn automatically. If necessary, a Child window will be clipped at the edges of its parent's client area.

A Child window can be the child of an App window or another Child window.

Unlike an App window, a Child window cannot have a title bar, system menu, maximise and minimise icons, or a menu bar, though it can have a single border and scroll-bars.

A Child window is created using the **CreateChildWin** subroutine.

The Client Area

The application has complete control over the appearance of a window's client area. Text, lines and rectangles may be drawn directly on the client area. However, should the client area be disturbed in any way (if, for instance, a dialog box is drawn in the client area) the application must restore it to its previous state.

Text Canvas

If required, the responsibility for maintaining the client area can be partially transferred to UIMS, by specifying that the window should have a text canvas. This is used to hold transient text strings drawn with **DrawTextString** in the window client area, so that they can be repainted at any time. If a window has a text canvas, the application does not have to redraw the text when the window is resized or uncovered by another contact. The default is for a window not to have a text canvas.

Notes:

1. The text canvas only stores the text strings and their positions in the client area; the appearance of the text is determined by the currently selected **Font** object. If the font is changed the appearance of the text will change when it is next redrawn.

2. Graphics shapes drawn with **DrawLine** and **DrawRect** are not stored in the text canvas. The application must ensure that these are redrawn when the window is updated.

3. The **Erase** subroutine can be used to clear the text canvas, and this also clears the whole of the client area. Note, however, that erasing all or part of the client area does not clear the text canvas – the stored text will be redrawn when the window is next updated.

Attributes

Style

The style of the window. This can be a combination of the following options:

- Display a horizontal scroll-bar.
- Display a vertical scroll-bar.
- Allow movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
- Text canvas (see above).

Subroutines – **ChildWinSetStyle**, **ChildWinGetStyle**.

Horizontal scroll-bar

The handle of the window's horizontal scroll-bar.

Subroutines – **ChildWinGetHScroll**.

Vertical scroll-bar

The handle of the window's vertical scroll-bar.

Subroutines – **ChildWinGetVScroll**.

Clip region

Defines a clipping region within the client area for all drawing operations. Text and graphics drawn outside the clipping region are not displayed.

Subroutines – **SetClip**, **GetClip**.

Drawrule

The handle of a **Drawrule** object used for all drawing operations within client area. This defines attributes such as foreground and background colours, text font, line width, etc. (see page 3-22).

Subroutines – **SetDrawrule**, **GetDrawrule**.

Pointer

The handle of a **Pointer** object used when the mouse pointer is within the window's client area.

Subroutines – **SetPointer**, **GetPointer**.

Cursor state

The type of cursor displayed in the client area and whether or not it is visible. The following types are available:

- Outline cursor (not supported on Microsoft Windows).
- Block cursor.

- Underline cursor.
- Vertical bar cursor.

Subroutines – **SetCursorState**, **GetCursorState**.

Cursor position	The position of the cursor relative to the origin (top left-hand corner) of the client area. The position is specified in text or graphics coordinates, depending on the coordinate mode of the application. Subroutines – **SetCursorPosition**, **GetCursorPosition**.
Children	The list of child contacts. Subroutines – **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.
Focus	The handle of the child contact which has the input focus. Subroutines – **SetContactFocus**, **GetChildFocus**.
Default button	The handle of the default **TitledButton** contact. This attribute is only applicable to windows with the **UIMS.WIN.DIALOG** style. Subroutines – **ChildWinSetDefButton**.

Common Contact Attributes

All the common contact attributes (see page 3-2) apply to **ChildWindow** contacts.

Other Subroutines

DrawTextString	Draws text on the client area or text canvas.
DrawLine	Draws a line on the client area.
DrawRect	Draws a rectangle on the client area.
Scroll	Scrolls the client area.
Erase	Erases a specified part of the client area, or the whole of the text canvas.

Clipboard

The **Clipboard** object provides access to the GUI system clipboard (if one is available). This allows the user to move data within an application and between UIMS applications and other applications running on the GUI.

Attributes

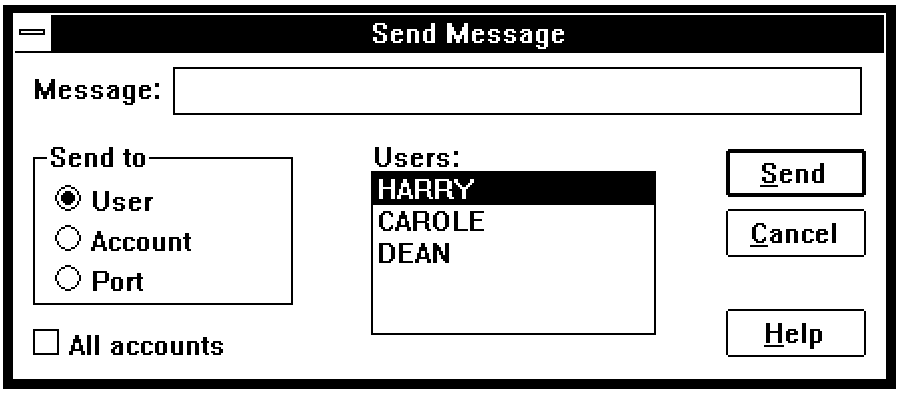
Content	The data on the clipboard. Subroutines – **ClipboardGetContent**, **ClipboardSetContent**, **Copy**, **Cut**, **Paste**
Size	The amount of data on the clipboard. When requesting the size of the clipboard contents, a format must be specified. If data is available, but it is not in the specified format, zero is returned. Subroutines – **ClipboardGetSize**.
Help index	A pointer to the related section of the application help file (see page 3-6). Subroutines – **SetHelpIndex**, **GetHelpIndex**.

Cut and Paste Operations

The **Copy**, **Cut** and **Paste** subroutines provide the means of transferring data between **EditBox** and **TextEditor** contacts and the clipboard. **Copy** and **Cut** place the selected text on the clipboard (in the case of **Cut,** removing it from the edit contact at the same time), while **Paste** inserts the contents of the clipboard into the edit contact at a specified position. Note, however, that **Cut**, **Copy** and **Paste** cannot be used with any other type of contact.

DialogBox

A **DialogBox** contact is a window that is used to prompt for information from the user. It does this by means of its child contacts (controls); typical dialog controls are check buttons, option buttons, edit boxes, list boxes and titled buttons.



When created, a dialog box is always application modal – the application will not continue until the user has responded to the dialog, but other applications continue to work normally. Two other modes are available: system modal and modeless. A system-modal dialog box disables the complete user interface; the user can do nothing until he has responded to the dialog. A modeless dialog box does not disable the parent window; the user can continue to work with the application while the dialog box is displayed.

A **DialogBox** contact is created with the **CreateDlgBox** subroutine.

Attributes

Mode

Modeless, application modal or system modal.

Subroutines – **DlgBoxSetMode**, **DlgBoxGetMode**.

Style

The style of the dialog box. This can be a combination of the following options:

- Movable – generates a title bar and a system menu with the Move command enabled.
- Closable – this is the same as movable, except that the Close command on the system menu is also enabled.

Subroutines – **DlgBoxSetStyle**, **DlgBoxGetStyle**.

Title	The text that will appear in the title bar. Note that a dialog box only has a title bar if it is movable. If there is no title bar, the title will not be displayed. Subroutines – **DlgBoxSetTitle**.
Children	The list of child contacts. Subroutines – **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.
Focus	The handle of the child contact which has the input focus. Subroutines – **SetContactFocus**, **GetChildFocus**.
Default button	The handle of the default **TitledButton** contact. Subroutines – **DlgBoxSetDefButton**.

Common Contact Attributes

All the common contact attributes (see page 3-2) except Border Style apply to **DialogBox** contacts.

Other Subroutines

DrawTextString	Draws text on the client area.
DrawLine	Draws a line on the client area.
DrawRect	Draws a rectangle on the client area.
Erase	Erases a specified part of the client area.

Display

A **Display** object provides access to the characteristics of a display or printer device. Its attributes can only be read, and on some platforms some attributes may not be supported.

Display objects are constructed by UIMS during initialisation, and whenever a new printer device is configured on the underlying GUI. The values are largely settings taken from the underlying GUI.

Note: Printer display objects are not supported on this version of UIMS. The subroutines concerned are provided for use on later releases.

Attributes

Pixel size	The dimensions in pixels of the display or the print area. Subroutines – **DisplayGetPixelSize**.

Other Subroutines

AppWinGetDisplay	Returns the handle of the **Display** object on which an App window is being displayed.
DisplayGetMetrics	Returns information about the sizes of the various window elements (title bar, border, etc.) when shown on the specified **Display** object.
GetDefaults	Returns the handles of the default **Display**, **Printer** and **TypeFace** objects.

Drawrule

A **Drawrule** object encapsulates the methods for drawing text and graphics in a window's client area.

UIMS provides a default **Drawrule**, the handle of which can be obtained by using **GetDrawrule** to fetch the handle of the drawrule for the Application context. Additional **Drawrule** objects can be created with the **CreateDrawrule** subroutine. The default **Brush**, **Font** and **Pen** objects for the application context will be attached to the newly-created drawrule. These can be changed with the appropriate subroutines (see below).

Drawrule Inheritance

There are two ways in which a drawrule becomes attached to a contact: by calling the **SetDrawrule** subroutine; or by inheritance from its parent:

A newly created contact inherits its parent's drawrule. This means that a contact created without a parent has no drawrule until it is either given a parent, or specifically given a drawrule with **SetDrawrule**.

Once a contact has a drawrule, it retains it until changed with **SetDrawrule**. However, a contact's drawrule can be removed by calling **SetDrawrule** and specifying a null handle. If the contact has a parent, the old drawrule will be replaced by that attached to the parent object. If the contact has no parent, the old drawrule will be removed and the contact will inherit a new drawrule when it is next attached to a parent object.

Attributes

Font	The handle of a **Font** object for character drawing (see page 3-30). Subroutines – **DrawruleSetFont**, **DrawruleGetFont**.
Pen	The handle of a **Pen** object for line drawing (see page 3-44). Subroutines – **DrawruleSetPen**, **DrawruleGetPen**.
Brush	The handle of a **Brush** object for area filling (see page 3-11). Subroutines – **DrawruleSetBrush**, **DrawruleGetBrush**.
Text mode	Character drawing mode. The following are available:

UIMS.TEXT.OPAQUE	Fill the text background with the selected background colour;
UIMS.TEXT.HOLLOW	Do not fill the text background.

Subroutines – **CreateDrawrule**.

Graphics mode

Graphics pen or brush drawing mode. The following are available:

UIMS.DRAW.CLEAR	Invert the Pen colour and combine the result with the colour on the screen by using a bit-wise AND operation.
UIMS.DRAW.COPY	Replace the colour on the screen with that of the selected Pen.
UIMS.DRAW.NOTCLEAR	Combine the Pen and screen colours with a bit-wise AND.
UIMS.DRAW.NOTCOPY	Replace the colour on the screen with the bit-wise inverse of the Pen colour.
UIMS.DRAW.NOTOR	Invert the Pen colour and combine the result with the colour on the screen by using a bit-wise OR operation.
UIMS.DRAW.NOTXOR	Combine the Pen and screen colours by means of a bit-wise exclusive-OR and then invert the result.
UIMS.DRAW.OR	Combine the Pen and screen colours by means of a bit-wise OR operation.
UIMS.DRAW.XOR	Combine the Pen and screen colours by means of a bit-wise exclusive-OR operation.

The effects of these graphics drawing modes are described in detail in Appendix B.

Subroutines – **CreateDrawrule**.

Colours

Foreground and background colours. These must be UIMS logical colours or RGB values. Note that the foreground colour is used only for text; the colours of lines and area fills are determined by the **Pen** and **Brush** objects respectively (see Figure 3-2).

Subroutines – **DrawruleSetColour**, **DrawruleGetColour**.

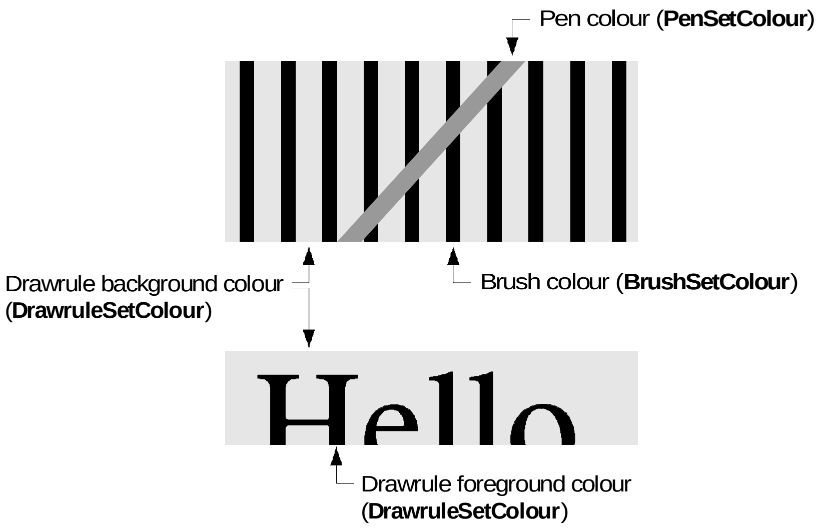

Pen colour (**PenSetColour**)
Drawrule background colour (**DrawruleSetColour**)
Brush colour (**BrushSetColour**)
Hello
Drawrule foreground colour (**DrawruleSetColour**)

Figure 3-2. Foreground and Background Colours

Other Subroutines

GetDrawrule	Returns the handle of the **Drawrule** that is attached to a specified object.
SetDrawrule	Attaches a new **Drawrule** to the specified object or contact.

EditBox

An **EditBox** contact is a single line text field in which text may be input and edited by the user, or presented for display by the application. The field can be left, right or decimal point aligned.

Within the edit box, the cursor can be moved with the keyboard or the mouse. Unless the Wait Pointer is on, when inside the edit box, the mouse pointer changes to a vertical I-beam; clicking within the edit box sets the cursor to the character position closest to the mouse pointer. Note that attempting to move the cursor beyond the edge of the field will automatically scroll the contents.

Text within the edit box may also be selected with the mouse or the keyboard. Making a selection generates a Select message which gives the start and end positions of the selected text; deselecting text generates a Select message with the start and end positions both given as zero.

Table 3-1 gives full details of the edit box keyboard and mouse interfaces.

An EditBox contact is created using the **CreateEditBox** subroutine.

Attributes

Content	The text displayed within the edit box. Subroutines – **EditBoxSetContent**, **EditBoxGetContent**.
Selection	The text which is highlighted within the edit box. Subroutines – **EditBoxSetSelected**.
Style	Whether or not the edit field is enclosed in a box. Subroutines – **CreateEditBox**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style and Drawrule apply to **EditBox** contacts.

User Interface

Table 3-1 summarises the mouse and keyboard interfaces for an edit box.

Table 3-1. User Interface for EditBox

Action	Result
Mouse Interface	
Single click	Positions the insertion point and drops the selection anchor.
Double click	Selects a word.
SHIFT+Single click	Positions the insertion point and extends the selection from the selection anchor to the insertion point.
Drag	Drops the selection anchor, moves the insertion point and extends the selection from the selection anchor to the insertion point.
Keyboard Interface	
LEFT ARROW, RIGHT ARROW	Removes the selection from any text and moves the insertion point in the indicated direction.
SHIFT+RIGHT ARROW, SHIFT+LEFT ARROW	Drops the selection anchor (if it is not already dropped), moves the insertion point and selects all text between the selection anchor and the insertion point.
CTRL+RIGHT ARROW, CTRL+LEFT ARROW	Moves the insertion point to the beginning of the word in the indicated direction.
SHIFT+CTRL+RIGHT ARROW, SHIFT+CTRL+LEFT ARROW	Drops the selection anchor (if it is not already dropped), moves the insertion point to the beginning of the word in the indicated direction, and selects all text between the selection anchor and the insertion point.
HOME	Removes the selection from any text and moves the insertion point to the beginning of the field.
SHIFT+HOME	Drops the selection anchor (if it is not already dropped), moves the insertion point to the beginning of the field, and selects all text between the selection anchor and the insertion point.
CTRL+HOME	As HOME.
SHIFT+CTRL+HOME	As SHIFT+HOME.
END	Removes the selection from any text and moves the insertion point to the end of the field.

(continued)

Table 3-1 User Interface for EditBox (continued)

Action	Result
SHIFT+END	Drops the selection anchor (if it is not already dropped), moves the insertion point to the end of the field, and selects all text between the selection anchor and the insertion point.
CTRL+END	As END.
SHIFT+CTRL+END	As SHIFT+END.
DELETE	If text is selected, deletes the text. Otherwise, deletes the character following the insertion point.
BACKSPACE	If text is selected, deletes the text. Otherwise, deletes the character preceding the insertion point.
SHIFT+DELETE	If text is selected, cuts the text to the clipboard. Otherwise, deletes the character following the insertion point.
SHIFT+INSERT	Pastes (inserts) the contents of the clipboard at the insertion point.
CTRL+INSERT	Copies the selected text to the clipboard, but does not delete it.

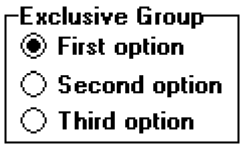
Note: When the user types a character, any selected text is automatically replaced by the character typed.

ExclusiveGroup

An **ExclusiveGroup** is a contact that manages a number of button contacts as a group. It has the following characteristics:

- Only one button in the group can be selected at a time.
- The group can be made up of **OptionButton** contacts only.

If required, the group may be given a heading and enclosed in a rectangle.



Note: The border and title of an **ExclusiveGroup** lie within its client area. Care must be taken when positioning option buttons, text and graphics, to ensure that they do not overwrite the border and the title.

An **ExclusiveGroup** is created using the **CreateExGroup** subroutine.

Attributes

Title	The text that will appear above the group. This will only be displayed if the group has a surrounding rectangle. Subroutines – **ExGroupSetTitle**.
Style	Whether or not the group has a surrounding rectangle. Subroutines – **ExGroupSetStyle**.
Selection	Which of the buttons in the group is currently selected. This is a read-only attribute; selection is made by the user, or by calling the appropriate option button subroutine. Subroutines – **ExGroupGetSel**, **OptionButtonSelect**, **OptionButtonDeselect**, **OptionButtonSetSelected**.
Children	The list of child **OptionButton** contacts. Subroutines – **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.

Focus	The handle of the child contact which has the input focus. Subroutines – **SetContactFocus**, **GetChildFocus**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to **ExclusiveGroup** contacts.

Font

A **Font** object defines the characteristics of the text font used when writing characters on a window's client area. A **Font** cannot be attached directly to a contact, but must be a child of the attached **Drawrule** object.

UIMS provides a default **Font**, the handle of which can be obtained by using **GetDrawrule** to fetch the handle of the drawrule for the Application context, and then calling the **DrawruleGetFont** subroutine. Additional **Font** objects can be created with the **CreateDrawFont** subroutine.

Attributes

TypeFace	The handle of a **TypeFace** object. Subroutines – **FontSetTypeFace**, **FontGetTypeFace**.
Style	A combination of the styles which are available in the selected typeface. Depending on the typeface, the following styles might be available: Normal, Bold, Italic, Outline, Underline, Strikeout. Subroutines – **FontSetStyle**, **FontGetStyle**.
Point size	The required point size for the font. A list of the point sizes available in the selected typeface may be obtained by calling the **TypeFaceGetPointSizes** subroutine. Subroutines – **FontSetPointSize**.
Font metrics	The dimensions, in pixels, of the selected style and size of the selected typeface, as follows: • The total height of the font – the ascent plus the descent (see below). • The height above the base line of the tallest characters (ascent). • The height of the longest descender (descent). • The distance between the descenders of one row of characters and the top of the tallest characters in the next row (leading). • The average width of the lower case characters. • The average width of the upper case characters. • The width of the widest character.

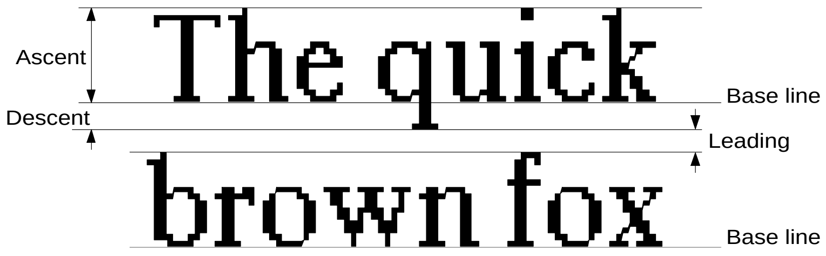


Figure 3-3. Font Metrics

These values are set when the font is created and cannot be changed by the programmer.

Subroutines – **FontGetMetrics**.

Other Subroutines

DrawruleGetFont

Returns the handle of the **Font** which is attached to the specified **Drawrule** object.

DrawruleSetFont

Attaches a **Font** to a **Drawrule** object.

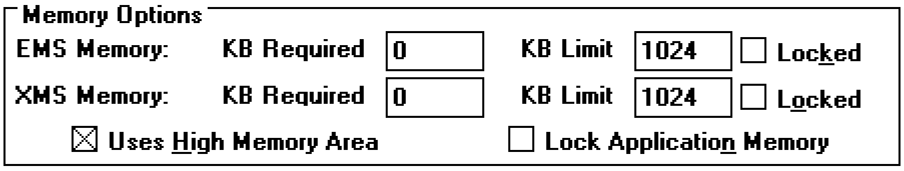
InclusiveGroup

An **InclusiveGroup** is a contact that manages a number of other contacts as a group. An inclusive group differs from an exclusive group in that it can contain contacts other than option buttons and that, if the group contains a number of buttons, more than one can be selected at once.

Except where used internally by a child contact, the cursor keys can be used to move the input focus within the group. The order in which contacts receive the focus depends on their positions in the list of children. Pressing TAB moves the focus to the next contact outside the group.

If required, the group may be given a heading and enclosed in a rectangle.

The example below shows an inclusive group containing **Text**, **EditBox** and **CheckButton** contacts. These are enclosed in a rectangle with a title.



Memory Options

EMS Memory: KB Required 0 KB Limit 1024 ☐ Locked

XMS Memory: KB Required 0 KB Limit 1024 ☐ Locked

☒ Uses High Memory Area ☐ Lock Application Memory

Note: The border and title of an **InclusiveGroup** lie within its client area. Care must be taken when positioning child contacts, text and graphics, to ensure that they do not overwrite the border or the title.

An **InclusiveGroup** is created using the **CreateIncGroup** subroutine.

Child Contacts

The following types of contact can be used within an inclusive group:

CheckButton,	**ListBox**,
ChildWindow,	**OptionButton**,
EditBox,	**Rectangle**,
ExclusiveGroup,	**ScrollBar**,
InclusiveGroup,	**Text**,
Line,	**TextEditor**.

Attributes

Title	The text that will appear above the group. This will only be displayed if the group has a surrounding rectangle. Subroutines – **IncGroupSetTitle**.

	Style	Whether or not the group has a surrounding rectangle. Subroutines – **IncGroupSetStyle**.
	Children	The list of child contacts. Subroutines – **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.
	Focus	The handle of the child contact which has the input focus. Subroutines – **SetContactFocus**, **GetChildFocus**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to **InclusiveGroup** contacts.

Other Subroutines

DrawTextString	Draws text on the client area.
DrawLine	Draws a line on the client area.
DrawRect	Draws a rectangle on the client area.
Erase	Erases a specified part of the client area.

Line

A **Line** contact provides a way of displaying a line within the client area of a window. The length and slope of the line are determined by the size and shape of an imaginary containing box. The line may be drawn with an arrowhead at either or both ends.

A **Line** contact will redraw or realign itself when required. Lines drawn directly onto the client area must be redrawn by the application.

Note: When a **Line** contact is created, its length and slope are determined by the positions of the two ends of the line. To change its size, the **Resize** subroutine must be used; when calling this, you must specify new values for the width and height of the containing box.

A **Line** contact is created using the **CreateLine** subroutine.

Attributes

Drawrule — The handle of an attached **Drawrule** object. This specifies the **Pen** object used to draw the line (see page 3-44). The **Drawrule** object is described on page 3-22.

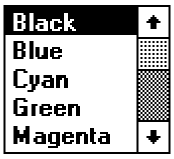
Subroutines – **SetDrawrule**, **GetDrawrule**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style and Event Mask apply to **Line** contacts.

ListBox

A **ListBox** contact allows the user to select one or more from a list of options. It consists of a box in which the available items are displayed as a vertical list. The list is displayed in the order supplied by the application; the **ListBox** contact cannot sort its contents. If there are more items available than will fit in the box, a vertical scroll-bar will automatically be attached to the box so that the user can scroll through the list. Scrolling is managed by the list box; the application does not have access to the scroll-bar attributes.



A list box can be configured so that the user can select only one item at a time, or to permit multiple selections. A selection is shown within the ListBox by highlighting the entire line. Tables 3-2 and 3-3 give details of the mouse and keyboard interfaces for standard and multiple-selection list boxes.

If required, a **ListBox** can be linked to an **EditBox** contact; when an item is selected it is copied into the edit box. The characters are inserted into the EditBox one at a time, as if they had been entered at the keyboard. The result will depend on whether or not each character passed any edit box validation mask criteria; the application is responsible for ensuring that the edit box will accept items copied from the list box.

A ListBox contact is created using the **CreateListBox** subroutine.

Attributes

Contents	The list of items (character strings). Subroutines – **ListBoxAddContent**, **ListBoxAddContents**, **ListBoxGetContent**, **ListBoxGetContents**, **ListBoxRemoveContent**, **ListBoxRemoveContents**.
Link	The handle of an **EditBox** contact to which the **ListBox** is linked. Subroutines – **ListBoxSetLink**.
Selections	The positions in the contents list of the items that are selected. Subroutines – **ListBoxAddSelection**, **ListBoxAddSelections**, **ListBoxGetSelections**, **ListBoxRemoveSelection**, **ListBoxRemoveSelections**.

Controls	Whether or not multiple selections are allowed.
	Subroutines – **CreateListBox**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to **ListBox** contacts.

User Interface

The tables below summarise the mouse and keyboard interfaces for standard and multiple-selection list boxes.

Table 3-2. User Interface for Standard List Box

Action	Result
Mouse Interface	
Single click	Selects the item and removes the selection from the previously selected item (if any).
Double click	Same as a single click.
Keyboard Interface	
SPACEBAR	Selects the item
RIGHT ARROW, DOWN ARROW	Selects the next item in the list and removes the selection from the previously selected item (if any).
LEFT ARROW, UP ARROW	Selects the preceding item in the list and removes the selection from the previously selected item (if any).
PAGE UP	Scrolls the currently selected item to the bottom of the list box, selects the first visible item in the list box, and removes the selection from the previously selected item (if any).
PAGE DOWN	Scrolls the currently selected item to the top of the list box, selects the last visible item in the list box, and removes the selection from the previously selected item (if any).
HOME	Scrolls the first item in the list to the top of the list box, selects the first item, and removes the selection from the previously selected item (if any).
END	Scrolls the last item in the list to the bottom of the list box, selects the last item, and removes the selection from the previously selected item (if any).

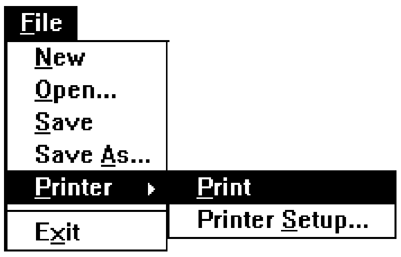
Table 3-3. User Interface for Multiple-selection List Box

Action	Result
Mouse Interface	
Single click	Toggles the selection state of the item, while preserving the selection state of all other items.
Double click	Same as a single click.
Keyboard Interface	
SPACEBAR	Toggles the selection state of the item, while preserving the selection state of all other items.
RIGHT ARROW, DOWN ARROW	Moves the list box cursor to the next item in the list.
LEFT ARROW, UP ARROW	Moves the list box cursor to the preceding item in the list.
PAGE UP	Scrolls the currently selected item to the bottom of the list box and moves the list box cursor to the first visible item in the list box.
PAGE DOWN	Scrolls the currently selected item to the top of the list box and moves the list box cursor to the last visible item in the list box.
HOME	Scrolls the first item in the list to the top of the list box and moves the list box cursor to the first item.
END	Scrolls the last item in the list to the bottom of the list box and moves the list box cursor to the last item.

Menu

A **Menu** contact consists of a vertical list of choices from which the user can select. The choices are **MenuItem** or **Menu** contacts. A menu can be used in two ways:

- It can be used as a pull-down menu, by attaching it to a **MenuBar** contact.
- It can be attached to another **Menu** contact to create a cascaded menu.



The parent **MenuBar** or **Menu** contact displays the title of the menu. The menu itself only appears when selected by the user.

A **Menu** contact is created by calling **CreatePullDownMenu** or **MakePullDownMenu**.

Attributes

Title	The text that will appear on the parent menu bar or menu. One of the characters in the title can be designated as a selector key by preceding it with an ampersand character. Subroutines – **MenuSetTitle**.
Children	A list containing the handles of child **MenuItem** and **Menu** contacts. Subroutines – **CreateMenuItem**, **CreatePullDownMenu**, **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.

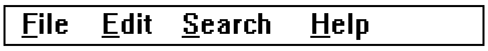
Common Contact Attributes

Of the common contact attributes, only the following apply to **Menu** contacts:

- Enabled/disabled state.
- Update display control.
- Event mask.

MenuBar

A **MenuBar** is a contact that consists of a horizontal list of choices displayed below the title of an **AppWindow** contact. It offers the first level of menu choice for a user. The choices a menu bar offers may be **Menu** or **MenuItem** contacts.



A **MenuBar** contact is created using the **CreateMenuBar** subroutine.

Attributes

Choices — A list of the handles of the menu bar's child **Menu** or **MenuItem** contacts.

Subroutines – **CreatePullDownMenu**, **CreateMenuItem**, **AddChild**, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetChild**, **GetChildren**, **GetChildCount**.

Common Contact Attributes

Of the common contact attributes, only the following apply to **MenuBar** contacts:

- Update display control.
- Event mask.

Other Subroutines

AppWinSetMenuBar
Attaches a menu bar to an **AppWindow** contact.

AppWinRemoveMenuBar
Removes the menu bar from an **AppWindow** contact.

AppWinGetMenuBar
Returns the handle of an App window's **MenuBar** contact, if any.

Comments

Keyboard access to the menu bar is by means of a selector key that is platform dependent and cannot be changed by the user. On Microsoft Windows, this selector is the ALT key.

MenuItem

A **MenuItem** contact allows the user to select an application command from a menu. It must be attached to a **Menu** or a **MenuBar** contact and consists of a title that appears on the parent contact. When the user selects a menu item, a button-press message is generated; this can be detected by the application, which must initiate the appropriate operation.

A **MenuItem** contact is created using the **CreateMenuItem** subroutine.

Attributes

Title	The text that will appear on the parent menu bar or menu. One of the characters in the title can be designated as a selector key by preceding it with an ampersand character. If a single hyphen is used as the title, a separator item is created. This appears as a continuous line across the width of its parent menu. Separator items cannot be selected by the user and should be used to visually group related menu items. Note that a separator item cannot be attached to a menu bar. Subroutines – **MenuItemSetTitle**.
Check mark	A mark (normally a tick or a cross, depending on the platform) that can be displayed beside a menu item to indicate that an option is selected. Subroutines – **MenuItemCheck**, **MenuItemUncheck**, **MenuItemSetCheckMark**, **MenuItemGetCheckMark**.
Autocheck	This is an operating mode that removes the burden of check mark control from the application. When selected, Autocheck automatically toggles the check mark on or off, as appropriate, each time the user selects the menu item. Subroutines – **MenuItemSetAutoCheck**.

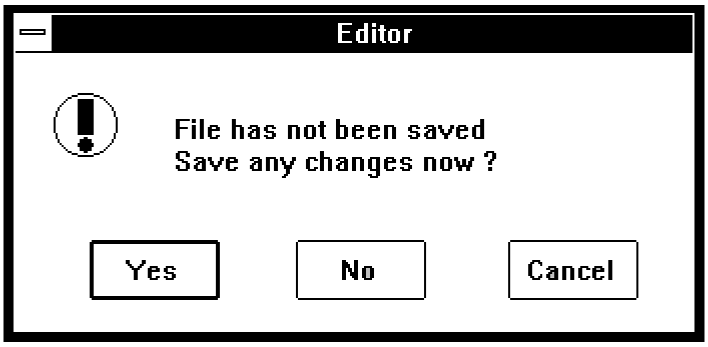
Common Contact Attributes

Of the common contact attributes, only the following apply to **Menu** contacts:

- Enabled/disabled state.
- Update display control.
- Event mask.

MessageBox

A **MessageBox** contact is a dialog box which displays a message and waits for the user to respond. It has up to three titled buttons and a graphic icon. A message box is always application modal (see page 3-19).



File has not been saved
Save any changes now ?

The programmer can define the icon displayed, the number of buttons, the titles on the buttons and the default button. The following icons are available:

Information:	ℹ
Warning:	❗
Alert:	🛑
Query:	❓

Alternatively a pre-defined style can be chosen; the following are available.

- An Information message box which displays a message and has one button. Unless changed, the button title is 'OK'.

- A Warning message box which displays a warning message and has two or three buttons. Unless changed, a two-button Warning box has 'OK' and 'Cancel' buttons and a three-button Warning box 'Yes', 'No' and 'Cancel' buttons.

- An Alert message box which displays a alert message and has two or three buttons. Unless changed, a two-button Alert box has 'Retry' and 'Cancel' buttons and a three-button Alert box 'Abort', 'Retry' and 'Cancel' buttons.

- A Query message box which displays a question mark and has two or three buttons. Unless changed, a two-button Query box has 'OK' and 'Cancel' buttons and a three-button Query box 'Yes', 'No' and 'Cancel' buttons.

The button titles can be changed if necessary, to suit the particular requirements of the application.

The size of the message box is adjusted according to the length of the message, which must be less than 200 characters long.

A **MessageBox** contact is created by calling the **CreateMessageBox** subroutine.

Attributes

Style	The icon, number of buttons and default button, or one of seven pre-defined styles: Information, Warning with two buttons, Warning with three buttons, Alert with two buttons, Alert with three buttons, Query with two buttons or Query with three buttons.
Title	The title to be displayed at the top of the message box.
Message	The message to be displayed.
Button Titles	A list of button titles, if required.

These are all set when the message box is created.

Common Contact Attributes

None of the common contact attributes apply to **MessageBox** contacts.

OptionButton

A **OptionButton** is a contact that allows the user to select one from a group of options. It consists of a small circle with a button title to the right. When the option is selected, the circle contains a mark of some kind – usually a second, filled in, circle in the centre of the button, though this depends on the platform.

Although option buttons can be used individually, they are normally grouped together in an ExclusiveGroup contact. When this is done, only one button in the group can be selected at a time. Exclusive groups of option buttons should be used to offer a number of mutually exclusive options.

A **OptionButton** contact is created using the **CreateOptionButton** subroutine.

Attributes

Title	The text that will appear beside the option button. One of the characters in the title can be designated as a selector key by preceding it with an ampersand character. Subroutines – **OptionButtonSetTitle**.
State	Whether or not the button is selected. Subroutines – **OptionButtonSelect**, **OptionButtonDeselect**, **OptionButtonSetSelected**, **OptionButtonGetSelected**.
Autotoggle	This is an operating mode that removes the burden of selection mark control from the application. When selected, Autotoggle automatically toggles the selection mark on or off, as appropriate, each time the user selects the button. Option buttons within an exclusive group always operate in Autotoggle mode. Subroutines – **OptionButtonSetToggle**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to **OptionButton** contacts.

Pen

A **Pen** object determines the appearance of lines drawn using the **Line** and **Rectangle** contacts, and the **DrawLine** and **DrawRect** subroutines. A **Pen** cannot be attached directly to a contact, but must be a child of a **Drawrule** object (see page 3-22).

UIMS provides a default **Pen**, the handle of which can be obtained by using **GetDrawrule** to fetch the handle of the drawrule for the Application context, and then calling the **DrawruleGetPen** subroutine. Additional **Pen** objects can be created using the **CreateDrawPen** subroutine.

Attributes

Colour

A UIMS logical colour or RGB value.

Subroutines – **PenSetColour**, **PenGetColour**.

Style

The style of the pen. The following styles are available:

UIMS.PEN.SOLID
A continuous line with colour and width as specified.

UIMS.PEN.HOLLOW
A transparent line. This is most useful when drawing rectangles – if the pen is continuous, the rectangle will be enclosed in a border; with a transparent pen, this border will be invisible.

Subroutines – **CreateDrawPen**.

Width

The width of the line in pixels.

Note: A pen width greater than zero can be inefficient on some display platforms.

Subroutines – **PenSetWidth**, **PenGetWidth**.

Other Subroutines

DrawruleGetPen
Returns the handle of the **Pen** which is attached to the specified **Drawrule** object.

DrawruleSetPen
Attaches a **Pen** to a **Drawrule** object.

Pointer

A **Pointer** object determines the shape and characteristics of the mouse pointer.

When the workstation has a mouse (or any other type of pointing device), the pointer shows the current location of the mouse. The pointer is automatically displayed and moved as the mouse is moved. If the workstation does not have a mouse, the pointer is not normally displayed.

The pointer is normally moved by the user but, if required, the application can control its position, or restrict it to a specific window. The type of pointer displayed is controlled partly by UIMS and partly by the application. The application can determine the type of pointer used within each window's client area, and can change it to the Wait Pointer type while processing takes place.

A **Pointer** object is created using the **CreatePointer** subroutine.

Pointer Inheritance

There are two ways in which a pointer becomes attached to a contact: by calling the **SetPointer** subroutine; or by inheritance from its parent.

A newly created contact inherits its parent's pointer. This means that a contact created without a parent has no pointer until it is either given a parent, or specifically given a pointer with **SetPointer**.

Once a contact has a pointer, it retains it until changed with **SetPointer**. However, a contact's pointer can be removed by calling **SetPointer** and specifying a null handle. If the contact has a parent, the old pointer will be replaced by that attached to the parent object. If the contact has no parent, the old pointer will be removed and the contact will inherit a new pointer when it is next attached to a parent object.

Attributes

Type

The shape of the pointer. This can be any of the following:

- Standard arrow pointer.
- Text I-beam pointer.
- Diagonal cross-hair pointer.
- Horizontal and vertical cross-hair pointer.
- Wait pointer – normally an hourglass.

Subroutines – **PointerSetType**, **PointerGetType**.

Other Subroutines

GetPointer	Returns the handle of the **Pointer** object that is attached to a specified object or contact.
GetPointerPos	Returns the pointer position, relative to either the screen or a specified contact.
GrabPointer	Traps the pointer within a contact.
SetPointer	Attaches a new **Pointer** object to a specified object or contact.
SetPointerPos	Sets the pointer position, relative to either the screen or a specified contact.
UngrabPointer	Releases the pointer if it has been trapped in a contact by **GrabPointer**.
WaitPointerOff	Changes the mouse pointer from the wait pointer to the pointer type specified by the **Pointer** object.
WaitPointerOn	Changes the mouse pointer to the wait pointer, overriding the pointer type specified by the **Pointer** object.

Rectangle

A **Rectangle** contact provides a way of displaying a rectangle within the client area of a window. It differs in the following ways from text drawn directly on a window's client area with the **DrawRect** subroutine:

- A **Rectangle** contact will redraw or realign itself when required. A rectangle drawn directly onto the client area must be redrawn by the application.
- The background of a **Rectangle** contact can be a different colour to that of the client area.

Note: When a **Rectangle** contact is created, its size is determined by the positions of its edges. To change its size, the **Resize** subroutine must be used; when calling this, you must specify new values for the width and height of the rectangle.

A **Rectangle** contact is created using the **CreateRect** subroutine.

Attributes

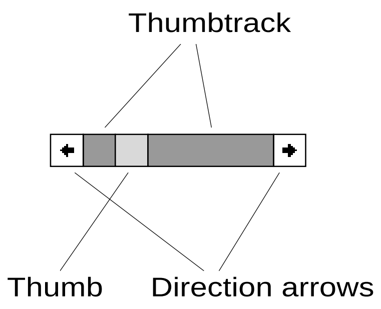
Style	Whether or not the Rectangle has a border. Subroutines – **CreateRect**.
Drawrule	The handle of an attached **Drawrule** object. This specifies the **Pen** object used to draw the rectangle (see page 3-44) and the **Brush** object used to fill the centre of the rectangle (see page 3-11). The **Drawrule** object is described on page 3-22. Subroutines – **SetDrawrule**, **GetDrawrule**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style and Event Mask apply to **Rectangle** contacts.

ScrollBar

A **ScrollBar** contact provides a graphical method of selecting one from a range of values. It consists of a band (or thumbtrack) with a small box containing an arrow at each end. A box on the thumbtrack (the thumb) acts as a slider which can be dragged along the thumbtrack using the mouse. The position of the thumb on the thumbtrack represents the currently selected value, in relation to the maximum and minimum values assigned to the end points of the thumbtrack.



Scroll-bars are most frequently seen in association with other contacts; they are provided in application and child windows, list boxes and text editors so that the user can view information which is not visible on the screen. A **ScrollBar** contact, however, is an independent control which can represent whatever the application requires. A minimum value is assigned to one end of the track and a maximum to the other; the user can then choose any value between these by simply moving the thumb. A minimum step can be specified to ensure that only meaningful values can be selected.

The user can move the thumb by dragging it with the mouse, or clicking either side of it, on the direction arrows or the thumbtrack. When a direction arrow is clicked, the thumb is moved by a small amount (line increment) in the appropriate direction; clicking the thumbtrack moves the thumb by a larger, page increment. Both the line and page increment can be set by the application. Table 3-4 on page 3-50 gives full details of the scroll-bar mouse and keyboard interfaces.

A **ScrollBar** can work in two modes: tracking and non-tracking. In tracking mode, each new thumb position is reported as it is moved; in non-tracking mode, the thumb position is only reported when the mouse button is released.

A **ScrollBar** contact is created using the **CreateScrollBar** subroutine.

Attributes

Type	Horizontal or Vertical.
	Subroutines – **CreateScrollBar**.
Tracking mode	Tracking or non-tracking.
	Subroutines – **ScrollBarSetTracking**.
Range	The maximum and minimum values represented by the ends of the thumbtrack.
	Subroutines – **ScrollBarSetRange**.
Thumb position	A value representing the thumb position, relative to the specified maximum and minimum values.
	Subroutines – **ScrollBarSetThumb**, **ScrollBarGetThumb**.
Increments	The line and page increments by which the thumb can move.
	Subroutines – **ScrollBarSetInc**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to **ScrollBar** contacts. Note, however, that none of the common contact attributes are applicable to scroll-bars created automatically as part of another contact.

Scroll-bar Messages

When the user operates a scroll-bar, a message is generated and a message returned to the application. The type of message depends on whether the scroll-bar is an independent control, or was created automatically as part of an App or Child window: **ScrollBar** contacts generate **UIMS.MSG.SCROLL** messages, while App and Child window horizontal and vertical scroll-bars respectively generate **UIMS.MSG.HSCROLL** and **UIMS.MSG.VSCROLL** messages.

When an application receives a scroll-bar message, the *Data2* parameter will contain a value that indicates what kind of scrolling is being performed. The application must use this information to determine how to position the scroll-bar thumb and what that position means to the application. Table 3-4 lists these *Data2* values and describes the user actions that generate them.

Table 3-4. User Interface for ScrollBar

Message *Data2* Value	Mouse	Keyboard
UIMS.SB.UP	User clicked the Up arrow on the scroll-bar.	User pressed the UP cursor key.
UIMS.SB.LEFT	User clicked the Left arrow on the scroll-bar.	User pressed the LEFT cursor key.
UIMS.SB.DOWN	User clicked the Down arrow on the scroll-bar.	User pressed the DOWN cursor key.
UIMS.SB.RIGHT	User clicked the Right arrow on the scroll-bar.	User pressed the RIGHT cursor key.
UIMS.SB.PAGEUP	User clicked the scroll-bar thumbtrack above the thumb.	User pressed the PAGEUP key.
UIMS.SB.PAGELEFT	User clicked the scroll-bar thumbtrack to the left of the thumb.	User pressed CTRL+PAGEUP.
UIMS.SB.PAGEDOWN	User clicked the scroll-bar thumbtrack below the thumb.	User pressed the PAGEDOWN key.
UIMS.SB.PAGERIGHT	User clicked the scroll-bar thumbtrack to the right of the thumb.	User pressed CTRL+PAGEDOWN.
UIMS.SB.THUMB	User has stopped dragging the thumb.	None
UIMS.SB.THUMBTRACK	User is dragging the thumb.	None

Speaker

The **Speaker** object provides access to the loudspeaker in the workstation or terminal.

Attributes

Pitch	The pitch (in Hertz) of the required sound.
Duration	The duration of the sound in milliseconds.
Repeats	The number of times to make the sound.
Delay	The time in milliseconds between repeats.

All of these are set using the **SoundSpeaker** subroutine.

SystemDictionary

The **SystemDictionary** object provides access to various workstation configuration values. It is constructed by UIMS during initialisation. The values are largely the default values for attributes of the underlying GUI. The **SystemDictionary** is UIMS system-wide and is accessed by all instances of all UIMS applications running on the workstation.

Attributes

Default screen	The handle of the default screen **Display** object (see page 3-20).
	Subroutines – **GetDefaults**.
Default printer	The handle of the default printer **Display** object (see page 3-20).
	Subroutines – **GetDefaults**.
Default typeface	The handle of the default **TypeFace** object (see page 3-59).
	Subroutines – **GetDefaults**.
Typefaces	A list of the available **TypeFace** objects (see page 3-59).
	Subroutines – **GetTypeFaces**, **GetTypeFace**.

Note: Printer display objects are not supported on this version of UIMS. The subroutines concerned are provided for use on later releases.

Text

A **Text** contact provides a way of displaying text within the client area of a window. It differs from text drawn directly on a window's client area with the **DrawTextString** subroutine in the following ways:

- A **Text** contact will redraw or realign itself when required. Text drawn directly onto the client area must be redrawn by the application.
- The text may be left or right aligned, justified or centred within the containing window boundary. Alignment of text within a window's client area is the responsibility of the application.
- The background of a **Text** contact can be a different colour to that of the client area.

A **Text** contact is created using the **CreateText** subroutine.

Attributes

Content	The text to be displayed. Subroutines – **TextSetContent**, **TextGetContent**.
Drawrule	The handle of an attached **Drawrule** object. This specifies the font used, the colour of the text, and the colour of the child window's background (refer to page 3-22 for details). Subroutines – **SetDrawrule**, **GetDrawrule**.
Alignment	Text alignment – left, right, both (justified) or centred. Subroutines – **TextSetJustification**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style and Event Mask apply to **Text** contacts.

TextEditor

A **TextEditor** contact is a text field in which text may be input and edited by the user, or presented for display by the application. It is similar to an **EditBox**, but allows the entry or display of more than one line of text.

The text within a **TextEditor** is divided into lines, each terminated by a carriage return. If the text editor contains more text than will fit into its display window, the text can be scrolled horizontally or vertically as necessary. If automatic scrolling is enabled, scrolling will take place as the cursor moves, or the mouse is dragged outside the contact. In addition, scroll-bars can be displayed so that the user can control text scrolling; these are managed entirely by the TextEditor and do not generate scroll messages.

Within the text editor, the cursor can be moved with the keyboard or the mouse. Unless the Wait Pointer is on, when inside the text editor, the mouse pointer changes to a vertical I-beam. Clicking within the text editor sets the cursor to the character position closest to the mouse pointer.

Text within the text editor may also be selected with the mouse or the keyboard. Making a selection generates a Select message which gives the start and end positions of the selected text; deselecting text generates a Select message with the start and end positions both given as zero.

Table 3-5 gives full details of the edit box keyboard and mouse interfaces.

A TextEditor contact is created using the **CreateTextEditor** subroutine.

Attributes

Style

The style of the text editor. This can be a combination of the following options:

- Border. If selected, the edit field is enclosed in a box.
- Display a horizontal scroll-bar.
- Display a vertical scroll-bar.
- Autoscroll. If selected, the text will scroll when the mouse is dragged outside the text editor.
- Read only. If selected, the text editor will be a display-only field, with no editing allowed.

Subroutines – **CreateTextEditor**.

Content	The text being edited or displayed. Subroutines – **TextEditorSetContent**, **TextEditorGetContent**, **TextEditorGetTextLen**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style and Drawrule apply to **TextEditor** contacts.

User Interface

The table below summarises the mouse and keyboard interfaces for a text editor.

Table 3-5. User Interface for Text Editor

Action	Result
Mouse Interface	
Single click	Positions the insertion point and drops the selection anchor.
Double click	Selects a word.
SHIFT+Single click	Positions the insertion point and extends the selection from the selection anchor to the insertion point.
Drag	Drops the selection anchor, moves the insertion point and extends the selection from the selection anchor to the insertion point.
Keyboard Interface	
Direction	Removes the selection from any text and moves the insertion point in the indicated direction.
SHIFT+Direction	Drops the selection anchor (if it is not already dropped), moves the insertion point and selects all text between the selection point and the insertion point.
CTRL+RIGHT ARROW, CTRL+LEFT ARROW	Moves the insertion point to the beginning of the word in the indicated direction.
SHIFT+CTRL+RIGHT ARROW, SHIFT+CTRL+LEFT ARROW	Drops the selection anchor (if it is not already dropped), moves the insertion point to the beginning of the word in the indicated direction, and selects all text between the selection anchor and the insertion point.
HOME	Removes the selection from any text and moves the insertion point to the beginning of the line.

(continued)

Table 3-5 User Interface for Text Editor (continued)

Action	Result
SHIFT+HOME	Drops the selection anchor (if it is not already dropped), moves the insertion point to the beginning of the line, and selects all text between the selection anchor and the insertion point.
CTRL+HOME	Places the cursor before the first character in the TextEditor.
SHIFT+CTRL+HOME	Drops the selection anchor (if it is not already dropped), places the cursor before the first character in the TextEditor, and selects all text between the selection anchor and the insertion point.
END	Removes the selection from any text and moves the insertion point to the end of the line.
SHIFT+END	Drops the selection anchor (if it is not already dropped), moves the insertion point to the end of the line, and selects all text between the selection anchor and the insertion point.
CTRL+END	Places the cursor after the last character in the TextEditor.
SHIFT+CTRL+END	Drops the selection anchor (if it is not already dropped), places the cursor after the last character in the TextEditor, and selects all text between the selection anchor and the insertion point.
DELETE	If text is selected, deletes the text. Otherwise, deletes the character following the insertion point.
BACKSPACE	If text is selected, deletes the text. Otherwise, deletes the character preceding the insertion point.
SHIFT+DELETE	If text is selected, cuts the text to the clipboard. Otherwise, deletes the character following the insertion point.
SHIFT+INSERT	Pastes (inserts) the contents of the clipboard at the insertion point.
CTRL+INSERT	Copies the selected text to the clipboard, but does not delete it.
PAGE UP	Scrolls the text up one line less than the height of the TextEditor.
CONTROL+PAGE UP	Scrolls the text left one character less than the width of the TextEditor.
PAGE DOWN	Scrolls the text down one line less than the height of the TextEditor.
CONTROL+PAGE DOWN	Scrolls the text right one character less than the width of the TextEditor.

(continued)

Table 3-5 User Interface for Text Editor (continued)

Action	Result
CTRL+ENTER	If the TextEditor is in a DialogBox, or in a window with dialog box style, ends the line and moves the cursor to the beginning of the next line.
CTRL+TAB	If the TextEditor is in a DialogBox, or in a window with dialog box style, inserts a tab character.

Note: When the user types a character, any selected text is automatically replaced by the character typed.

TitledButton

A **TitledButton** contact is a push button that is used to initiate an action. For example, a dialog box will normally contain a button with the legend 'OK'; when the user clicks on this button, the contents of the dialog box will be returned to the application for processing.

The button is displayed with a rectangular border enclosing a text caption or graphic image. If required, the border of the button can be shown thickened; this is used in dialog boxes to indicate the default action (see page 3-19).

A TitledButton contact is created using the **CreateTitledButton** subroutine. It has the following attributes, which can set or read by using the appropriate subroutines.

Attributes

Title

The text that will appear inside the button, or the name of a file containing a graphic image. Note, however, that an image can only be attached to a TitledButton when it is created (with **CreateTitledButton**) and that, once attached, it cannot be changed.

Subroutines – **TitledButtonSetTitle**.

Style

The style can be one of the following:

- Draw a normal (thin) border round the button.
- Draw a thickened border round the button.

Notes:

1. Within a **DialogBox** contact, the default button has a thickened border.
2. If a TitledButton is created containing an image its style cannot be changed.

Subroutines – **TitledButtonSetStyle**, **DlgBoxSetDefButton**, **AppWinSetDefButton**, **ChildWinSetDefButton**.

Common Contact Attributes

All common contact attributes (see page 3-2) except Border Style apply to a **TitledButton** contact.

TypeFace

TypeFace objects are created by UIMS from the typefaces available on the display platform.

A typeface is a set of characters (letters, numerals, punctuation marks and symbols) that share a common design and character set. Each **TypeFace** object consists of a group of typefaces that have similar stroke width and serif characteristics; in most cases a range of point sizes and styles (bold, italic, etc.) will be available.

Note: The terms UIMS uses to describe fonts, typefaces and families of fonts do not necessarily correspond to traditional typographic terms.

Type Styles

Each typeface will also be available in one or more styles: normal, bold, italic, outline, underline and strikeout. Figure 3-4 illustrates these different styles in the Helvetica font.

This is a line of Helvetica

This is a line of Helvetica bold

This is a line of Helvetica italic

This is a line of Helvetica outline *

This is a line of Helvetica underline

~~This is a line of Helvetica strikeout~~

* For illustration only. Outline is not normally available in the Helvetica typeface.

Figure 3-4. TypeFace Styles

Where a particular style is available for the typeface concerned, UIMS will use it; otherwise UIMS will try to synthesise the style. If the style cannot be easily be synthesised, the nearest equivalent will be selected.

Note: Some styles are particularly difficult to synthesise. In particular, outline cannot generally be used unless the typeface concerned includes an outline style. Similarly, for some typefaces, it may not be possible to use strikeout style.

TypeFaces and Fonts

A typeface is made available to the application by attaching it to a **Font** object and selecting a style (Bold, Italic, etc.) and point size from those that are available. The same typeface can be attached to several different fonts.

The handles of the available typefaces can be obtained by calling the **GetTypeFace** and **GetTypeFaces** subroutines.

Attributes

Name	The name of the typeface (Times Roman, Helvetica, etc.). Subroutines – **TypeFaceGetName**.
Point Sizes	The available point sizes. Subroutines – **TypeFaceGetPointSize**, **TypeFaceGetPointSizes**.

General Subroutines

This section lists UIMS subroutines that have not been mentioned elsewhere in this chapter.

Management Subroutines

GetMsg	Retrieves the next message in the message queue for the session.
AddTimer	Creates a timer which will generate a timer message when the timer expires.
RemoveTimer	Removes a timer created with **AddTimer**.
SetTeWindow	Changes the window that is used as the application's 'terminal emulation' (TE) window – that is the window in which output printed to the terminal (using PRINT, CRT, etc.) will be displayed.

Object-Wide Subroutines

Destroy	Destroys an object or contact.
GetObjectParent	Returns the parent of an object.

NewView Subroutines

NewView is described in detail in Chapter 5.

CreateNVContactGroup	Creates a NewView contact group.
CreateNVHotspotGroup	Creates a group of NewView hot-spots within the application's terminal emulation window.
DestroyNVGroup	Destroys a NewView group created with **CreateNVContactGroup** or **CreateNVHotspotGroup**.
SetEnabledNVGroup	Enables or disables a NewView group.
SetMappedNVGroup	Allows you to decide whether or not a NewView group is displayed on the screen.

ReMapNVLine25
Allows you to use a UIMS message box to display system messages which the host sends to line 25 of the terminal screen.

ChangeNVContacts
Changes the response strings generated by contacts in a NewView group.

ChangeNVButtonGroup
Changes the titles of the buttons in a NewView button group and the response strings generated by them. It can also be used to control whether or not buttons in the group are visible.

DDE Subroutines

DDE.PEEK Uses a dynamic-data exchange (DDE) conversation to request data from a WIndows application.

DDE.POKE Uses a DDE conversation to send data to a WIndows application.

DDE.EXECUTE
Uses a DDE conversation to send a command or commands to a WIndows application.

DDE.OPENADVISE
Establishes a 'permanent' DDE link to a Windows application.

DDE.ADVISE Obtains data from a permanent DDE link.

DDE.CLOSEADVISE
Closes a permanent DDE link.

Image Management Subroutines

StartImage Loads the image manager.

DisplayImage Displays an image in a specified window.

EraseImage Removes a displayed image.

StopImage Unloads the image manager.

Other Subroutines

InitialiseUims Initialises the UIMS environment.

SignOn Signs on a UIMS session and creates an **AppContext** object for the new session.

SignOff Signs off a UIMS session.

GetUimsVersion	Returns the UIMS version number and revision level.
SetSync	Switches between synchronous and asynchronous UIMS function call error response handling.
GetErrorText	Returns the text associated with a specified error code.
BitTest	Returns the state of a specified element in a composite value.
HiByte	Returns the value of the most-significant byte of a word (2 byte) value.
LoByte	Returns the value of the least-significant byte of a word (2 byte) value.
Execute	Starts a program on the PC.
SystemCommand	Runs a DOS system command on the PC.
SendKeys	Sends a sequence of key-presses to the active Windows application, as if they had been typed at the keyboard.
SetUimsMode	Restores message processing after NewView and application control subroutines, and DATA/BASIC commands that send data to or receive data from the terminal.

Chapter 4
Messages

This chapter describes how a UIMS application uses messages to receive user input. It also lists the different types of message and gives details of their parameters.

Overview

Every mouse or keyboard operation the user makes in using an application triggers a UIMS event. When an event occurs, a message is generated which is initially directed to the contact which currently has the focus. This then passes to its parent, which passes it to its parent, and so on until it reaches the application.

Message Loop

An essential part of any UIMS application is a message loop, containing a call to the **GetMsg** subroutine – this fetches messages as they are passed to the application and thus allows the application to respond to user actions. The **GetMsg** subroutine requires ten parameters, as follows:

- A number representing how long (in tenths of a second) to wait for an message to occur. This can allow an application to perform a background task while waiting for an event to occur. If zero is specified, **GetMsg** will not return until a message is received.
- Variables in which to return the handles of the application context, window and contact in which the event occurred.
- A variable in which to return the type of message.
- A variable in which to return a number representing the time the event occurred. This is only valid for certain types of message.
- Four variables in which to return additional message-specific parameters.

Message processing is best organised as a series of embedded case statements, with each level switching on a different message parameter. You are recommended to switch first on the window in which the event occurred, and then on the type of message. You can then, if necessary, test for the specific contact. It is unlikely that you will need to test the application context, as very few applications will have more than one.

A message loop has the following basic structure:

```
Until (user wants to exit) do
    Fetch the next message
    Process message
Loop
```

A simple message loop is shown in the following example:

```
USER.WANTS.TO.EXIT = FALSE
LOOP UNTIL USER.WANTS.TO.EXIT DO

    CALL GetMsg(0, CONTEXT, WINDOW, CONTACT, MSGTYPE, TIMESTAMP,
                  DATA1, DATA2, DATA3, DATA4)

    BEGIN CASE

    CASE WINDOW = WIN1
        BEGIN CASE
        CASE MSGTYPE = UIMS.MSG.MENUITEM
            GOSUB HANDLE.WIN1.MENUS

        CASE MSGTYPE = UIMS.MSG.EXIT
            USER.WANTS.TO.EXIT = TRUE

        END CASE
    END CASE
REPEAT
```

Note that you only need to process those messages which directly affect your application – in this case **UIMS.MSG.MENUITEM** and **UIMS.MSG.EXIT** messages. All others can be ignored. The subroutine HANDLE.WIN1.MENU should test for the selected menu item.

Not all messages reach the application. At any stage in the propagation process, an object may process the message – the result may be to convert one type of message into another or to simply not pass the message on. An example of this occurs when the user clicks on a button contact – the message generated is initially a mouse click message, but this is converted by the button contact into a button press message.

Masking Messages

A UIMS application can generate a great many messages which it does not need to process. In particular, every time the mouse is moved, one or more pointer motion messages are generated.

In order that the application should not be swamped with messages in which it is not interested, UIMS provides an Event Mask mechanism which allows the programmer to choose which types of message will be received by the application. An event mask can be applied to the application as a whole, through the **AppContext**, or to individual contacts. If a message is disabled at a contact, this does not prevent messages reaching the contact, but stops them being passed on to its parent.

An event mask is set by calling the **SetEventMask** subroutine. A mask is constructed by adding together the individual masks for the types of message you wish to receive. For example:

```
MASK = UIMS.EM.BUTTONPRESS+UIMS.EM.KEYPRESS
CALL SetEventMask(CONTACT, MASK, ERR)
```

You can find out which types of message are enabled by calling the **GetEventMask** subroutine. Note, however, if you need to change an existing mask, you cannot simply add or subtract an individual mask – you must first check whether or not the individual mask is already set. For example:

```
* First fetch the current event mask
CALL GetEventMask(CONTACT, MASK)

* Then pass the result to BitTest to find out if MENUITEM
* messages are enabled
CALL BitTest(STYLE, UIMS.EM.MENUITEM, ENABLED)

* If they are not already enabled, enable them
IF NOT(ENABLED) THEN
    MASK = MASK + UIMS.EM.MENUITEM
    CALL SetEventMask(CONTACT, MASK, ERR)
END
```

Default Event Masks

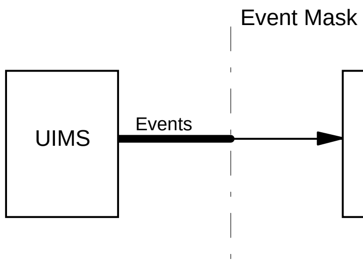
When a UIMS application is started, the following messages are enabled at the application context: **UIMS.MSG.BUTTONPRESS**, **UIMS.MSG.CLOSE**, **UIMS.MSG.EXIT**, **UIMS.MSG.KEYPRESS**, **UIMS.MSG.MENUITEM**, **UIMS.MSG.NOTIFY**.

All newly created objects and contacts have all message types enabled, except **UIMS.MSG.IDLE**.

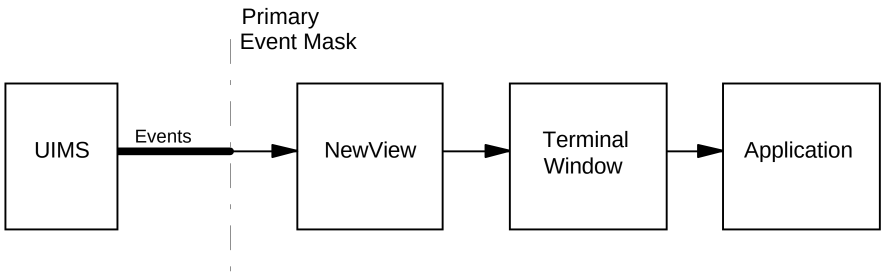
Secondary Event Mask

In addition to the event masks described above, the application context has a secondary event mask. This is normally only required in applications which do not have a message loop, such as NewView applications.

In a normal UIMS application with a message loop, the relationship between UIMS, the application and the application's event masks is as follows:

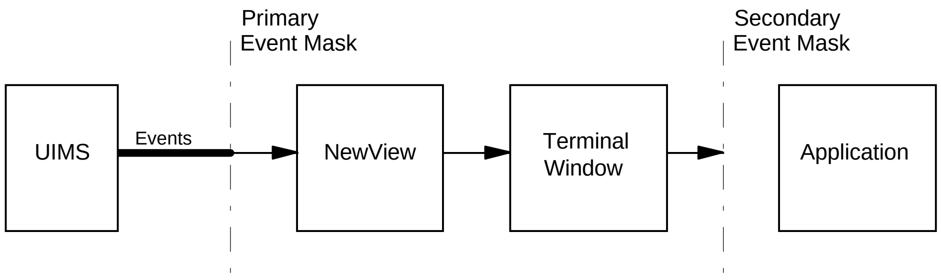


When using NewView, however, there are two additional components in the system: NewView itself, and the terminal window.



In this case, the primary event mask (that set for the application context using **SetEventMask**) determines which types of message should be passed to NewView. A pre-defined NewView event mask, **UIMS.EM.NEWVIEW**, should be used to ensure that NewView receives the correct types of message. Many of the messages are processed by NewView, but some are passed on to the terminal window, and these can in turn be passed on to the application. If the application has no message loop to process these, they will be interpreted as text to be displayed, and will be printed in the terminal window.

The secondary event mask overcomes this problem by masking-out these messages before they reach the application, as follows:



To set a secondary event mask, use the **SetSecondaryEventMask** subroutine. The following example disables all types of message:

```
SECMASK = 0        ;* disable all messages
NONMASK = FALSE    ;* disable non-maskable messages
CALL SetSecondaryEventMask(CONTEXT, SECMASK, NONMASK, FALSE, ERR)
```

The second parameter to **SetSecondaryEventMask** is an event mask with the same format as used for **SetEventMask**, while the third controls **UIMS.MSG.CREATE** and **UIMS.MSG.DESTROY** messages, which cannot be disabled with a normal event mask.

The fourth parameter is provided for use in future versions of UIMS. Its value will be ignored.

If you are writing a full UIMS application with a message loop, you will not normally need to set a secondary event mask. The default setting enables all maskable messages, and disables **UIMS.MSG.CREATE** and **UIMS.MSG.DESTROY** messages. You can find out the current setting of the secondary event mask by calling the **GetSecondaryEventMask** subroutine.

Message Categories

Messages can be grouped in to six categories:

Keyboard Messages

A keyboard message is generated is generated whenever the user presses a key on the keyboard.

Message types – **UIMS.MSG.KEYPRESS**.

Focus Messages

Focus messages are generated when the input focus changes from contact to contact. Keyboard messages are always passed initially to the contact which has the focus. Focus messages should be used by the application to initiate housekeeping tasks, such as displaying or removing a text cursor.

Message types – **UIMS.MSG.ENTER**, **UIMS.MSG.LEAVE**.

Pointer Messages

Pointer messages are generated when the mouse is moved and when the mouse buttons are pressed and released. Note that there are two levels of pointer messages: at the lower level (motion, press and release), every separate mouse move, and button press and release is reported; at the higher level (click, double-click and drag), some pre-processing is carried out in order to simplify an application's message handling.

Message types – **UIMS.MSG.CLICK**, **UIMS.MSG.DBLCLICK**, **UIMS.MSG.DRAG**, **UIMS.MSG.MOTION**, **UIMS.MSG.PRESS**, **UIMS.MSG.RELEASE**.

Window Messages

A window message is generated when the state of an App or Child window changes.

Message types – **UIMS.MSG.CLOSE**, **UIMS.MSG.CREATE**, **UIMS.MSG.DESTROY**, **UIMS.MSG.HSCROLL**, **UIMS.MSG.KILL**, **UIMS.MSG.MOVE**, **UIMS.MSG.SIZE**, **UIMS.MSG.UPDATE**, **UIMS.MSG.VSCROLL**.

Control Messages

Control messages are generated by user operation of contacts.

Message types – **UIMS.MSG.BUTTONPRESS**, **UIMS.MSG.LBOX.DESELECT, UIMS.MSG.LBOX.SELECT**, **UIMS.MSG.MENUITEM**, **UIMS.MSG.SCROLL**, **UIMS.MSG.SELECT**.

Application Messages

These are general messages which are not connected with any particular window or contact. They report such occurrences as errors and requests to close the application.

Message types – **UIMS.MSG.EXIT**, **UIMS.MSG.IDLE**, **UIMS.MSG.NOTIFY**, **UIMS.MSG.TIMER**.

Message Descriptions

The sections which follow list the UIMS messages in alphabetical order. Each description includes details of the conditions under which the message is generated, the value of the message code, any message-specific parameters returned by **GetMsg**, the corresponding event mask and any additional information.

The descriptions refer to the **GetMsg** parameters (*vContact*, *vData1*, *vData2*, etc.) in which message data is returned. These parameter names are the same as those given in the description of the **GetMsg** subroutine in Chapter 6.

Parameters

The following basic parameters are common to all UIMS messages:

- The message type (*vMsgType*). This determines the format of any message-specific data.
- The event context (*vContext*) – the handle of the application context in which the event occurred.
- The event window (*vWindow*) – the handle of the window in which the event occurred.
- The event contact (*vContact*) – the handle of the contact in which the event occurred.
- A time stamp (*vTimeStamp*) – this is a number which gives some indication of the order in which events occurred. At present, this information is only returned by pointer messages.

In addition, there are four parameters (*vData1* to *vData4*) which return message-specific data.

In many cases the event window and the event contact will be the same. Note, however, that for some types of message, the event context, window and/or contact may not be meaningful.

UIMS.MSG.BUTTONPRESS

A button press message is generated when the user operates a **TitledButton**, **OptionButton** or **CheckButton** contact.

Value

24

Message-specific Parameters

None.

Event Mask

UIMS.EM.BUTTONPRESS

UIMS.MSG.CLICK

This type of message is generated when a Release event follows a Press event in the same contact, with no intervening Motion events.

Value

5

Message-specific Parameters

vData1 The horizontal coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

Note: The value returned in *vData1* is offset by 65536. To obtain the true value, use the following code:

vData1 = INT(*vData1* / 65536)

vData2 The vertical coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

vData3 This contains the states of any mouse buttons which have not changed, and the states of the keyboard modifier keys (SHIFT, CTRL, ALT, etc.). The value returned is a combination of the pointer and keyboard modifier states listed in Appendix A.

vData4 The number of the mouse button which has been clicked. Note that the values produced by the different mouse button combinations are hardware dependent.

Event Mask

UIMS.EM.CLICK

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

An indication of the time at which the event occurred is given by a value returned in the *vTimeStamp* parameter.

UIMS.MSG.CLOSE

A Close message is generated when the user closes a window.

Value

11

Message-specific Parameters

None.

Event Mask

UIMS.EM.CLOSE

Comments

This message asks the application to close the specified window – this can be done by calling the **Destroy** subroutine or, if preferred, by using **UnMap** to make the window invisible.

A Close message may be followed by additional messages.

Note that when the user closes the application's Root window, an Exit message is generated instead of a Close message.

UIMS.MSG.CREATE

A Create message is generated when an App window is created.

Value	97
Message-specific Parameters	None.
Event Mask	None.
Comments	Create messages are normally disabled, but can be enabled by using the **SetSecondaryEventMask** subroutine – see page 4-4 for details. Create messages are not generated when other types of contact are created.

UIMS.MSG.DBLCLICK

A Double click message is generated when two Click events occur in the same contact, with no intervening Motion events and within the multi-click period set for the GUI platform.

Value

6

Message-specific Parameters

vData1 The horizontal coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

Note: The value returned in *vData1* is offset by 65536. To obtain the true value, use the following code:

vData1 = INT(*vData1* / 65536)

vData2 The vertical coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

vData3 This contains the states of any mouse buttons which have not changed, and the states of the modifier keys (SHIFT, CTRL, ALT, etc.). The value returned is a combination of the pointer and key modifier states listed in Appendix A.

vData4 The number of the mouse button which has been double-clicked. Note that the values produced by the different mouse button combinations are hardware dependent.

Event Mask

UIMS.EM.DBLCLICK

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

An indication of the time at which the event occurred is given by a value returned in the *vTimeStamp* parameter.

If the second click occurs after the multi-click period has expired, separate Click or Press and Release messages will be generated.

Enabling Double-click messages also enables Click messages.

UIMS.MSG.DESTROY

A Destroy message is generated when an App window is destroyed.

Value	98
Message-specific Parameters	None.
Event Mask	None.
Comments	Destroy messages are normally disabled, but can be enabled by using the **SetSecondaryEventMask** subroutine – see page 4-4 for details. Destroy messages are not generated when other types of contact are destroyed.

UIMS.MSG.DRAG

There are two types of Drag message:

- A drag-start message is generated when a primary button (button 1) Press event is followed immediately by a Motion event in the same contact.
- A drag-end message occurs when a Motion event with the drag and button 1 modifiers set is followed by a button 1 release in the same contact.

Value

8

Message-specific Parameters

vData1	The horizontal coordinate of the pointer location, relative to the left-hand edge of the event window's client area. **Note:** The value returned in *vData1* is offset by 65536. To obtain the true value, use the following code: *vData1* = INT(*vData1* / 65536)
vData2	The vertical coordinate of the pointer location, relative to the left-hand edge of the event window's client area.
vData3	This contains the states of any mouse buttons which have not changed, and the states of the modifier keys (SHIFT, CTRL, ALT, etc.). The value returned is a combination of the pointer and key modifier states listed in Appendix A. The presence of the **UIK.P.DRAG** pointer modifier indicates a drag-start message. If this modifier is not present, the message results from ending a drag operation.
vData4	Always set to 1.

Event Mask

UIMS.EM.DRAG

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

An indication of the time at which the event occurred is given by a value returned in the *vTimeStamp* parameter.

A drag-start message is always be preceded by a Press message in the same contact.

Drag messages are generated for the primary button (button 1) only. Drag operations with other buttons must be identified by means of the Press, Release and Motion pointer messages.

UIMS.MSG.ENTER

This message is generated when the input focus is passed to a contact. The event contact is the contact that is receiving the focus.

Value

1

Message-specific Parameters

None.

Event Mask

UIMS.EM.ENTER

Comments

The *vContact* parameter returns the handle of the contact which is receiving the focus.

UIMS.MSG.EXIT

An Exit message is generated when the user closes the application.

Value

16

Message-specific Parameters

None.

Event Mask

UIMS.EM.EXIT

Comments

This message asks the application to close itself down. It should be used to initiate house-keeping tasks such as saving un-saved documents.

Note that this message will be generated if UIMS runs out of resources. Under these circumstances it may not be possible to display any dialogs which request confirmation from the user.

UIMS.MSG.HSCROLL

This message is generated when the user operates any of the controls on an App or Child window's horizontal scroll bar.

Value

15

Message-specific Parameters

vData1	Not applicable.
vData2	The scroll bar operation. This will be one of the following values.

UIMS.SB.LEFT	The user clicked the scroll-bar Left arrow.
UIMS.SB.RIGHT	The user clicked the scroll-bar Right arrow.
UIMS.SB.PAGELEFT	The user clicked the scroll-bar thumb-track to the left of the thumb.
UIMS.SB.PAGERIGHT	The user clicked the scroll-bar thumb-track to the right of the thumb.
UIMS.SB.THUMB	The user has stopped dragging the thumb.
UIMS.SB.THUMBTRACK	The user is dragging the thumb.

vData3	Not applicable.
vData4	A value representing the new thumb position.

Event Mask

UIMS.EM.HSCROLL

Comments

On some display platforms, thumb-track scroll messages may not be generated.

This message is only generated when the user operates a horizontal scroll bar which forms part of an App or Child window. Operating a horizontal **ScrollBar** contact generates **UIMS.MSG.SCROLL** messages.

UIMS.MSG.IDLE

An Idle message is generated by UIMS when there are no events to report.

Value

19

Message-specific Parameters

None.

Event Mask

UIMS.EM.IDLE

Comments

Idle messages are initially sent to the application context, with the result that the *vWindow* and *vContact* parameters will be **NULL**. The application's message loop should be written to allow for this.

This message can only be enabled for the **AppContext** object.

This message can be used by the application to control the background processing of lengthy tasks.

UIMS.MSG.KEYPRESS

This type of message is generated whenever the user presses a key on the keyboard.

Value

9

Message-specific Parameters

vData1	The keyboard modifier state (SHIFT, CTRL and ALT key states).
vData2	The virtual key code of the key (see Appendix A).
vData3, *vData4*	Unused (returned set to zero).

Event Mask

UIMS.EM.KEYPRESS

Comments

The *vContact* parameter returns the handle of the contact which had the input focus at the time the key was pressed.

UIMS.MSG.KILL

A Kill message is generated when a contact ceases to exist.

Value

12

Message-specific Parameters

None.

Event Mask

UIMS.EM.KILL

Comments

An application can destroy a specified contact by calling the **Destroy** subroutine. Note, however, that if the contact has any children, these will also be destroyed.

UIMS.MSG.LBOX.DESELECT

A list-box deselect message is generated when a selected item in the list is deselected.

Value

26

Message-specific Parameters

vData2	The position of the deselected item within the list box. The list is numbered starting from zero.
vData1, *vData3*, *vData4*	Not applicable.

Event Mask

UIMS.EM.LBOX.DESELECT

UIMS.MSG.LBOX.SELECT

A list-box select message is generated when an item is selected from the list.

Value

25

Message-specific Parameters

vData2	The position of the selected item within the list box. The list is numbered starting from zero.
vData1, *vData3*, *vData4*	Not applicable.

Event Mask

UIMS.EM.LBOX.SELECT

UIMS.MSG.LEAVE

This type of message is generated when a contact loses the input focus. The event contact is the contact that is losing the focus.

Value	2
Message-specific Parameters	None.
Event Mask	**UIMS.EM.LEAVE**
Comments	The *vContact* parameter returns the handle of the contact which is losing the focus.

UIMS.MSG.MENUITEM

This type of message is generated when an item on a menu or menu bar is selected.

Value	21
Message-specific Parameters	None.
Event Mask	**UIMS.EM.MENUITEM**

UIMS.MSG.MOTION

This type of message is generated whenever the pointer is moved. The number of Motion messages generated for a given amount of movement may vary, since this depends on hardware interrupts. However, an application which has requested Motion messages is guaranteed at least one Motion message whenever the pointer moves and comes to rest.

Value

7

Message-specific Parameters

vData1 The horizontal coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

Note: The value returned in *vData1* is offset by 65536. To obtain the true value, use the following code:

vData1 = INT(*vData1* / 65536)

vData2 The vertical coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

vData3 This contains the states of the mouse buttons and the keyboard modifier keys (SHIFT, CTRL, ALT, etc.). The value returned is a combination of the pointer and keyboard modifier states listed in Appendix A.

vData4 Always zero.

Event Mask

UIMS.EM.MOTION

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

An indication of the time at which the event occurred is given by a value returned in the *vTimeStamp* parameter.

If the pointer has been constrained with the **GrabPointer** subroutine, Motion messages are generated periodically, even if the pointer does not move.

UIMS.MSG.MOVE

A Move message is generated when a contact is moved, either by the user, or by the application.

Value

27

Message-specific Parameters

vData1	The horizontal coordinate of the contact's new position in coordinate units.
vData2	The vertical coordinate of the contact's new position in coordinate units.
vData3	The overall width of the contact in coordinate units.
vData4	The overall height of the contact in coordinate units.

Note: The values returned in *vData1* and *vData3* are offset by 65536. To obtain the true values, use the following code:

vData1 = INT(*vData1* / 65536)
vData3 = INT(*vData3* / 65536)

Event Mask

UIMS.EM.MOVE

Comments

The values returned in *vData1* and *vData2* specify the position of the top left-hand corner of the contact, relative to the top left-hand corner of its parent's client area (position 0,0). Note, however, that for contacts that are children of the application context, the position returned is relative to the top, left-hand corner of the display (position 0,0).

The values returned in *vData1*, *vData2*, *vData3* and *vData4* will depend on the coordinate mode (text or graphics) currently selected for the application context.

Note: Contacts that can be moved by the user can be positioned to the nearest pixel, whichever coordinate mode is selected. In text mode, therefore, the values returned by a **UIMS.MSG.MOVE** message are accurate only to the nearest character position.

UIMS.MSG.NOTIFY

This type of message is generated when UIMS wishes to notify the application of an error. In particular, notify messages are used in asynchronous error mode to inform the application of errors which in synchronous mode would be returned in the subroutines' *vErr* parameters.

Value

17

Message-specific Parameters

vData1	Not applicable.
vData2	The name of the subroutine in which the error occurred.
vData3	Not applicable.
vData4	The error number.

Event Mask

UIMS.EM.NOTIFY

Comments

Notify messages are always sent directly to the application, with the result that the *vContext*, *vWindow* and *vContact* parameters are returned set to **NULL**.

A textual description of the error can be obtained by calling the **GetErrorText** subroutine.

UIMS.MSG.PRESS

This type of message is generated when one of the buttons on the mouse is pressed.

Value

3

Message-specific Parameters

vData1 The horizontal coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

Note: The value returned in *vData1* is offset by 65536. To obtain the true value, use the following code:

vData1 = INT(*vData1* / 65536)

vData2 The vertical coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

vData3 This contains the states of any mouse buttons which have not changed, and the states of the keyboard modifier keys (SHIFT, CTRL, ALT, etc.). The value returned is a combination of the pointer and keyboard modifier states listed in Appendix A.

vData4 The number of the mouse button which has been pressed. Note that the values produced by the different mouse button combinations are hardware dependent.

Event Mask

UIMS.EM.PRESS

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

An indication of the time at which the event occurred is given by a value returned in the *vTimeStamp* parameter.

It should not be assumed that a Press message will be followed by a Release message, unless the pointer has been constrained with the **GrabPointer** subroutine. This is because the release could occur in a different contact which might consume the message (for example, if a dialog box is popped up on a Press event, the release might occur in the dialog box). If the release occurs in another application, the Release event will not be reported.

UIMS.MSG.RELEASE

This type of message is generated when a mouse button is released.

Value

4

Message-specific Parameters

vData1 The horizontal coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

Note: The value returned in *vData1* is offset by 65536. To obtain the true value, use the following code:

vData1 = INT(*vData1* / 65536)

vData2 The vertical coordinate of the pointer location, relative to the left-hand edge of the event window's client area.

vData3 This contains the states of any mouse buttons which have not changed, and the states of the keyboard modifier keys (SHIFT, CTRL, ALT, etc.). The value returned is a combination of the pointer and keyboard modifier states listed in Appendix A.

vData4 The number of the mouse button which was released. Note that the values produced by the different mouse button combinations are hardware dependent.

Event Mask

UIMS.EM.RELEASE

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

An indication of the time at which the event occurred is given by a value returned in the *vTimeStamp* parameter.

It should not be assumed that a Press message will be followed by a Release message, unless the pointer has been constrained with the **GrabPointer** subroutine. This is because the release could occur in a different contact which might consume the message (for example, if a dialog box is popped up on a Press event, the release might occur in the dialog box). If the release occurs in another application, the Release event will not be reported.

UIMS.MSG.SCROLL

This type of message is generated when the user operates any of the controls on a **ScrollBar** contact.

Value

22

Message-specific Parameters

vData1 Not applicable.

vData2 The scroll bar operation. This will be one of the following values.

UIMS.SB.LEFT	The user clicked the scroll-bar Left arrow (horizontal scroll bar).
UIMS.SB.UP	The user clicked the scroll-bar Up arrow (vertical scroll bar).
UIMS.SB.RIGHT	The user clicked the scroll-bar Right arrow (horizontal scroll bar).
UIMS.SB.DOWN	The user clicked the scroll-bar Down arrow (vertical scroll bar).
UIMS.SB.PAGELEFT	The user clicked the scroll-bar thumb-track to the left of the thumb (horizontal scroll bar).
UIMS.SB.PAGEUP	The user clicked the scroll-bar thumb-track above the thumb (vertical scroll bar).
UIMS.SB.PAGERIGHT	The user clicked the scroll-bar thumb-track to the right of the thumb (horizontal scroll bar).
UIMS.SB.PAGEDOWN	The user clicked the scroll-bar thumb-track below the thumb (vertical scroll bar).
UIMS.SB.THUMB	The user has stopped dragging the thumb.
UIMS.SB.THUMBTRACK	The user is dragging the thumb.

vData3 Not applicable.

vData4 A value representing the new thumb position.

Event Mask

UIMS.EM.SCROLL

Comments

On some display platforms, thumb-track scroll messages may not be generated.

This message is only generated when the user operates a **ScrollBar** contact. Operating a scroll bars which forms part of an App or Child window generates **UIMS.MSG.HSCROLL** or **UIMS.MSG.VSCROLL** messages as appropriate.

UIMS.MSG.SELECT

This type of message is generated when text or a graphic object is selected.

Value

23

Message-specific Parameters

The message-specific parameters returned in a Select message depend on the type of contact in which the select operation occurred. At present the only contacts that can receive select messages are the **EditBox** and **TextEditor**, and the only data type that may be selected is text.

Note: The values returned in *vData1* and *vData3* are offset by 65536. To obtain the true values, use the following code:

vData1 = INT(*vData1* / 65536)
vData3 = INT(*vData3* / 65536)

vData1	The number of the line containing the start position.
vData2	The character number of the start position, within the line specified in *vData1*. The first character in the line is numbered zero.
vData3	The number of the line containing the end position.
vData4	The number of the character following the end position, within the line specified in *vData3*.

The first line in a **TextEditor** is numbered 0. For an **EditBox** contact, *vData1* and *vData3* are always zero.

Deselecting text generates a Select message with all four data parameters set to zero.

Event Mask

UIMS.EM.SELECT

UIMS.MSG.SIZE

A Size message is generated when a contact is changed in size, either by the user, or by the application.

Value

13

Message-specific Parameters

vData1 The new width of the contact's client area in coordinate units.

Note that the value returned in *vData1* is offset by 65536. To obtain the true value, use the following:

vData1 = INT(*vData1* / 65536)

vData2 The new height of the contact's client area in coordinate units.

vData4 The state of the window. This will be one of the following values:

UIMS.WS.MAX	The window has been maximised.
UIMS.WS.MIN	The window has been minimised.
UIMS.WS.NORMAL	The window has been resized, but has not been maximised or minimised.

Event Mask

UIMS.EM.SIZE

Comments

The values returned in *vData1* and *vData2* will depend on the coordinate mode (text or graphics) currently selected for the application context.

Note: Contacts that can be changed in size by the user can be sized to the nearest pixel, whichever coordinate mode is selected. In text mode, therefore, the values returned in *vData1* and *vData2* are accurate only to the nearest character position.

UIMS.MSG.TIMER

A timer message is generated by UIMS when a timeout value specified by the application has expired.

Value

18

Message-specific Parameters

None.

Event Mask

UIMS.EM.TIMER

Comments

The *vContact* parameter returns the handle of the timer.

For a Timer message the *vWindow* parameter will be **NULL**. The application's message loop should be written to allow for this.

UIMS.MSG.UPDATE

An Update message is generated when part or all of a contact becomes exposed – this usually occurs when the contact is made visible or when another contact is moved. The exposed region of the contact is divided into non-overlapping rectangles and an Update message is generated for each. Several Update messages may be generated as the result of a single user action.

Value

10

Message-specific Parameters

vData1	The position of the left-hand edge of the exposed region, relative to the left-hand edge of the contact's client area.
vData2	The position of the top edge of the exposed region, relative to the top edge of the contact's client area.
vData3	The position of the right-hand edge of the exposed region, relative to the left-hand edge of the contact's client area.
vData4	The position of the bottom edge of the exposed region, relative to the top edge of the contact's client area.

Note: The values returned in *vData1* and *vData3* are offset by 65536. To obtain the true values, use the following code:

vData1 = INT(*vData1* / 65536)
vData3 = INT(*vData3* / 65536)

Event Mask

UIMS.EM.UPDATE

Comments

The values returned in *vData1*, *vData2*, *vData3* and *vData4* will depend on the coordinate mode (text or graphics) currently selected for the application context.

The application will normally only receive Update messages for App and Child windows. Update messages for other types of contact are processed by the contact concerned.

UIMS.MSG.VSCROLL

This type of message is generated when the user operates any of the controls on an App or Child window's vertical scroll bar.

Value

14

Message-specific Parameters

vData1	Not applicable.
vData2	The scroll bar operation. This will be one of the following values.

UIMS.SB.UP	The user clicked the scroll-bar Up arrow.
UIMS.SB.DOWN	The user clicked the scroll-bar Down arrow.
UIMS.SB.PAGEUP	The user clicked the scroll-bar thumb-track above the thumb.
UIMS.SB.PAGEDOWN	The user clicked the scroll-bar thumb-track below the thumb.
UIMS.SB.THUMB	The user has stopped dragging the thumb.
UIMS.SB.THUMBTRACK	The user is dragging the thumb.

vData3	Not applicable.
vData4	A value representing the new thumb position.

Event Mask

UIMS.EM.VSCROLL

Comments

On some display platforms, thumb-track scroll messages may not be generated.

This message is only generated when the user operates a vertical scroll bar which forms part of an App or Child window. Operating a vertical **ScrollBar** contact generates **UIMS.MSG.SCROLL** messages.

Chapter 5
NewView

This chapter describes the UIMS NewView subsystem for enhancing existing applications.

Introduction

NewView allows existing character applications to be easily converted so that the user can use a mouse in addition to the normal keyboard interface. There are two ways in which this can be done:

- Certain types of contact can be made to generate text when clicked with the mouse. This text is passed to application as if it had been typed at the keyboard.

  For example, if to save a file, the user must type the character F and then press RETURN, a button with the title 'Save' might be created and set up to generate F followed by carriage return. The user could then click the button to save the file as an alternative to using the keyboard.

- Areas of the screen can be designated as 'hot-spots'. These also generate text when clicked with the mouse.

  For example, an application might display a menu consisting of three items, each selected by typing a letter. With NewView, the screen area containing each menu item could be set up as a hot-spot and made to generate the corresponding letter. The user could then select an item from this menu by simply pointing to the item required and clicking with the mouse.

  The user can identify hot-spots by the shape of the mouse pointer; when pointing to a hot-spot, it changes to a hand shape.

Assigning Text Strings

Text strings are assigned to contacts and hot-spots by creating NewView groups. In the case of hot-spots, defining a group also sets the sizes and positions of the hot-spots. A hot-spot group would normally be needed for each screen displayed by an application, while a single contact group could be shared by several screens. Groups can be enabled and disabled according to which screen is displayed.

The text strings assigned to the contacts in a group can be changed if necessary as required by different application screens. Similarly, the titles of the contacts can be changed at any time (by calling the appropriate UIMS subroutine).

The Terminal Window

While hot-spots can be set up in the RealLink window, any NewView contacts are likely to obscure the text displayed by the application. It is therefore recommended that an application which uses NewView contacts should create its own application window and a separate child window to act as the terminal window. If the child window is made smaller than the application window, the unused parts of its client area can be used for buttons. In addition, the AppWindow can be given application-specific menus.

Running NewView Applications on Normal Terminals

A NewView application can be written so that the NewView features are only used when running on RealLink. This means that only a single version of each application is needed on the host.

NewView Groups

Only the following types of contact can be used in NewView groups:

MenuItem
TitledButton

The following types of contact can also be used in NewView applications, but cannot be used in groups:

AppWindow
ChildWindow
MenuBar
Menu
Text
Line
Rectangle

NewView contacts can be created within an application, or compiled on the PC from a resource script (see Chapter 7) and loaded using the **LoadAppRes** subroutine. The use of compiled resources will minimise changes to the application; alternatively, a separate cataloged DATA/BASIC subroutine could be used to create the contacts.

RealLink uses graphics coordinate mode internally – NewView applications must therefore be set into this mode (by calling the **SetCoordMode** subroutine and specifying **UIMS.COORD.GRAPHIC**) before any UIMS resources are created. Note, however. that the size and position of a hot-spot is always specified in character positions.

Controlling the State of a Group

The state of a NewView group can be controlled by two subroutines: **SetEnabledNVGroup** and **SetMappedNVGroup**. These set all the contacts or hot-spots in the group to the same state.

When a group is created, its state is set to that of the first contact or hot-spot in the group. Note, however, that the states of the individual contacts are not changed. It is therefore important to ensure that all contacts in a group are initially in the same enabled and mapped state.

NewView Subroutines

The following subroutines are available to create and control NewView groups:

CreateNVContactGroup
Creates a NewView contact group.

CreateNVHotspotGroup
Creates a group of NewView hot-spots within the application's terminal emulation window.

ChangeNVContacts
Changes the response strings generated by contacts in a NewView group.

ChangeNVButtonGroup
Changes the titles of the buttons in a NewView button group and the response strings generated by them. It can also be used to control whether or not buttons in the group are visible.

DestroyNVGroup
Destroys a NewView group created with **CreateNVContactGroup** or **CreateNVHotspotGroup**.

SetEnabledNVGroup
Enables or disables a NewView group.

SetMappedNVGroup
Allows you to decide whether or not a NewView group is displayed on the screen.

These subroutines are described in detail in Chapter 6.

Setting the Terminal Window

All output that is printed to the terminal by an application (using PRINT, CRT, etc.) is displayed in the terminal window. In addition, NewView hot-spots are always defined as areas within this window.

The terminal window is normally the RealLink window, but it can be changed, if required, to an AppWindow or ChildWindow created by the application, by calling the subroutine **SetTeWindow**. If a child window is used, it must have an AppWindow as its parent, but it can be made smaller than the AppWindow, leaving room for other contacts.

Notes:

1. At present it is not possible to automatically change the size of a child when its parent is changed. It is therefore recommended that, if a child window is used as the terminal window in this way, its parent AppWindow should not be sizable.

2. The application is responsible for returning the terminal window to the RealLink window on exit. If this is not done, RealLink will be unable to continue, and an Unrecoverable Application Error may occur.

Menus

An AppWindow created by the application, whether used as the terminal window or not, can be given application-specific menus. These must be created in the same way as the menus in a UIMS application:

- A **MenuBar** must be created and attached to the **AppWindow**.

- **Menu** contacts must be created and made children of the **MenuBar**.

- **MenuItem** contacts must be created and made children of the appropriate menus. The items can be of two types:

   1. Application-specific menu items which form part of a NewView contact group, and which return text strings to the application.

   2. RealLink print, edit and help menu items. The items listed in Table 5-1 are available.

      If a menu item is created with one of the identifiers listed in the table, it will have the same function as the corresponding RealLink menu item. It can, however, be attached to any menu, or to the menu bar, and it can be given a different title if required.

Table 5-1. NewView RealLink Menu Items

Menu	Menu Item	Identifier/Handle
File	Print Selection	**ID.FILEPRINT**
File	Printer Setup	**ID.FILEPRINTERSETUP**
File	Print Window	**ID.FILEPRINTWINDOW**
Edit	Copy	**ID.EDITCOPY**
Edit	Paste	**ID.EDITPASTE**
Edit	Copy Window	**ID.EDITCOPYWINDOW**
Help	Index	**ID.HELPINDEX**
Help	Commands	**ID.HELPCOMMANDS**
Help	Keyboard	**ID.HELPKEYBOARD**
Help	Application	**ID.HELPAPP**

These constants are defined in the item RFWDEFS in the file UIMS-TOOLS. This item must be included at beginning of your application.

Note: If you use the Resource Compiler to create your menus, you will need to include these definitions in your resource script. This can be done by coping RFWDEFS onto your PC using one of the RealLink file transfer utilities (LanFTU or HOST-WS – see the *RealLink for Windows User Manual* for details); you can then use a #include command to incorporate the contents of the file. You must give the include file on your PC the extension '.H'.

The menu bar, menus and menu items can be created by calling the appropriate UIMS subroutines from within the application, or on the PC by compiling a resource script (see Chapter 7).

System Messages

If required, system messages that the host sends to line 25 of a normal terminal can be redirected to a UIMS message box by calling the **ReMapNVLine25** subroutine. This should not be done, however, in applications which use line 25 for a continuous display of status information.

On-line Help

If required you can create a Help file specific to your NewView application as described in Chapter 8. This file can then be loaded using the **SetNVHelp** subroutine and displayed by giving the user access to the **ID.HELPAPP** menu item (see page 5-6 for details).

A NewView Application

The following details the steps that must be added to an application, so that it can use NewView.

1. INCLUDE statements which specify the RealLink and UIMS constant definitions:

   ```
   INCLUDE RFWDEFS FROM UIMS-TOOLS
   INCLUDE UIMSDEFS FROM UIMS-TOOLS
   INCLUDE UIMSCOMMON FROM UIMS-TOOLS
   ```

   The second and third of these are not required if only hot-spots are being used.

2. A call to the **InitialiseUims** subroutine. This is not required if only hot-spots are being used.

   Once this call has been made, the common variable **UIMS.CAPABLE** can be tested to determine whether the application is running on RealLink or on a normal terminal. The remaining steps must only be carried out if running on RealLink.

3. A call to the **SignOn** subroutine. This is not required if only hot-spots are being used.

4. A call to the **SetEventMask** subroutine, specifying **UIMS.EM.NEWVIEW** as the new event mask.

5. A call to the **SetCoordMode** subroutine, specifying **UIMS.COORD.GRAPHIC** as the coordinate mode.

6. Subroutine calls to create the UIMS resources (windows, buttons, menu items and other contacts). To minimise changes to the application, these could be located in a separate cataloged subroutine, or loaded with **LoadAppRes** from a compiled resource script on the PC.

7. If a window other than the RealLink window is to be used as the terminal window, a call to **SetTeWindow** will be required.

8. If you have written a help file for your application and have created a menu item to display it, you must call the **SetNVHelp** subroutine to load the help file.

9. Subroutine calls to create NewView contact and hot-spot groups.

10. Each time the application displays a different screen, the appropriate NewView groups must be enabled and disabled by calls to **SetMappedNVGroup** and **SetEnabledNVGroup**.

11. When the application terminates, if it is running on RealLink, the following must be done:

    - Use **DestroyNVGroup** to destroy all NewView contact and hot-spot groups.

    - If a window other than the RealLink window has been used as the terminal window, call **SetTeWindow** to return the terminal window to RealLink (see Chapter 6 for details). This must be done before signing off from UIMS.

    - Call the **SignOff** subroutine to sign off from UIMS. This is not required if the application has not signed on to UIMS.

Chapter 6
Subroutine Reference

This chapter describes each of the UIMS DATA/BASIC subroutines in detail. They are listed in alphabetical order, with related routines grouped together.

Introduction

Each UIMS and NewView subroutine must be called as an external cataloged DATA/BASIC subroutine with the CALL command; for example:

```
CALL SetEnabled(CONTEXT, EDIT.PASTE, TRUE, ERR)
```

Because DATA/BASIC is case sensitive, the subroutine names must be typed exactly as shown in the syntax descriptions. Using the wrong case for even one letter will result in a fatal error at run time and entry to the DATA/BASIC debugger. Note that there will be no visual indication of this unless either the RealLink window is visible, or you have set the Terminal window to your own App or Child window; you can, however, return to the RealLink window by pressing the Restore key (refer to the *RealLink for Windows User Manual* for details).

Include Items

The UIMS and NewView constants are defined in the file UIMS-TOOLS. There are four items in this file which must be included at the beginning of your application. These are:

UIMSDEFS	Defines constants and error messages.
UIMSCOMMON	Declares COMMON variables.

These items will be required for most applications, but can be omitted if the application uses only NewView hot-spots and the **Execute**, **SystemCommand** and **SendKeys** subroutines.

RFWDEFS	Defines constants and error messages for NewView applications, and the **Execute**, **SystemCommand** and **SendKeys** subroutines. Only required if these features are used in the application.
RFWKEYS	Contains key definitions for the **SendKeys** subroutine. Only required if **SendKeys** is used in the application.
UIMS-DDE	Contains definitions for the Dynamic Data Exchange (DDE) subroutines. Only required if DDE is used in the application.

Numeric Parameters

All numeric parameters must be passed as integer values. If a value which includes a decimal point is used, this will be converted to zero.

Returned Values

In most cases, when a UIMS or NewView subroutine is called, a result is returned – a completion code, for example, or the states of one or more attributes. Since DATA/BASIC does not support user-defined functions, in all cases the programmer must supply one or more variables in which to return these values. The parameters in which results are returned are indicated in the subroutine descriptions by a lower case 'v' prefixing the parameter name; for example, *vDisplay*.

Errors

UIMS can handle errors in two ways: synchronously or asynchronously. The **SetSync** subroutine is used to select the required mode. The default is asynchronous.

Asynchronous Error Handling

In asynchronous mode, errors are handled as follows:

- Unless otherwise stated, subroutines which return only a completion status code return immediately. The value returned in *vErr* is always zero (**UIMS.SUCCESS**).
- If a subroutine which creates an object is passed a non-zero identifier (*Ident* parameter), the subroutine returns immediately; the handle returned will be set to the value of the supplied identifier. If the identifier is zero, errors are handled synchronously (see above).
- All other subroutines do not return until completion. Any value(s) returned should be checked for validity.

In asynchronous mode, if an error occurs, a **UIMS.MSG.NOTIFY** message is generated (see Chapter 4 for details). This should be processed by the application's message loop in the same way as other types of message.

Synchronous Error Handling

In synchronous mode, errors are handled as follows:

- Subroutines which return a completion status code do not return until it is known whether the call was successful. If an error has occurred, the error code is returned in the *vErr* parameter.
- Subroutines which create objects do not return until the object has been created; if an error occurs, a null handle is returned.
- All other subroutines do not return until completion. Any value(s) returned should be checked for validity.

Note: Some subroutines always return errors synchronously. This is mentioned in the descriptions of the subroutines concerned.

AddChild, AddChildren

These subroutines attach children to an object.

- **AddChild** adds a single child.
- **AddChildren** adds a number of children.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AddChild(*Context***,** *Object***,** *Index***,** *Child***,** *vErr***)**

CALL AddChildren(*Context***,** *Object***,** *Index***,** *aChildren***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Object	The handle of the object to which you wish to add the children.
Index	The point in the list of children at which the new child or children are to be added. The list is numbered starting from 0 and new entries are added before the specified existing entry. An index of -1 adds the new entry to the end of the list.
Child	The handle of the contact that is to be made a child of the object.
aChildren	A dynamic array containing the handles of the contacts that are to be made children of the object.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

When **AddChild** or **AddChildren** are called, the objects added will be drawn immediately, provided the objects concerned are mappable and the parent is currently displayed.

If only one child is being added to an object, **AddChild** is faster than **AddChildren**.

See Also

GetChild, **GetChildren**, **RemoveChild**, **RemoveChildren**, **GetObjectParent**.

AddTimer

This subroutine creates a timer and sets it running.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AddTimer(*Context***,** *Interval***,** *vHandle***)**

Syntax Elements

Context	The handle of the application context.
Interval	The time in milliseconds between timer messages.
vHandle	A variable in which to return a handle to the newly created timer.

Comments

Each time the timer expires a **UIMS.MSG.TIMER** message is generated.

The timer created runs repeatedly until removed with the **RemoveTimer** subroutine. If a one-shot timer is required it must be removed after the first timer message.

See Also

RemoveTimer.

AppHelp

This subroutine displays application help text.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AppHelp(*Context***,** *Section***,** *vErr***)**

Syntax Elements

Context	The handle of the **AppContext**.
Section	The help-id of the required section of the help file. If this parameter is 0, the index will be displayed.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

SetHelpFile, **GetHelpFile**, **SetHelpIndex**, **GetHelpIndex**.

AppWinGetDisplay – AppWinGetVScroll

These subroutines return the different attributes of an **AppWindow** contact.

- **AppWinGetDisplay** returns the handle of the screen on which the App window is being displayed.
- **AppWinGetHScroll** returns the handle of the App window's horizontal scroll-bar, if any.
- **AppWinGetMenuBar** returns the handle of an App window's **MenuBar** contact, if any.
- **AppWinGetStyle** returns the style of the App window.
- **AppWinGetVScroll** returns the handle of the App window's vertical scroll-bar, if any.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AppWinGetDisplay(*Context***,** *AppWindow***,** *vDisplay***)**

CALL AppWinGetHScroll(*Context***,** *AppWindow***,** *vHScrollBar***)**

CALL AppWinGetMenuBar(*Context***,** *AppWindow***,** *vMenuBar***)**

CALL AppWinGetStyle(*Context***,** *AppWindow***,** *vWinStyle***)**

CALL AppWinGetVScroll(*Context***,** *AppWindow***,** *vVScrollBar***)**

Syntax Elements

Context	The handle of the application context.
AppWindow	The handle of the **AppWindow** contact.
vDisplay	A variable in which to return the handle of the **Display** object on which the window is being shown.
vHScrollBar	A variable in which to return the handle of the window's horizontal scroll-bar. If zero is returned, the window either does not have a horizontal scroll-bar or its horizontal scroll-bar is hidden.

See **CreateAppWin** for a more detailed description of App window scroll-bars.

vMenuBar A variable in which to return the handle of the window's menu bar. If zero is returned, the window does not have a menu bar.

vWinStyle A variable in which a value representing the style of the window will be returned. This value will be a combination of one or more of the following:

UIMS.WIN.CLOSABLE	The window can be closed by the user.
UIMS.WIN.DIALOG	Permits movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
UIMS.WIN.HSCROLL	The window has a horizontal scroll-bar.
UIMS.WIN.ICONISABLE	The window has a minimise box.
UIMS.WIN.MOVABLE	The window can be moved by the user.
UIMS.WIN.SIZABLE	The size of the window can be changed by the user.
UIMS.WIN.TEXT	The window has a text canvas attached.
UIMS.WIN.VSCROLL	The window has a vertical scroll-bar.

The **BitTest** subroutine can be used to test the individual elements which make up the returned value.

See **CreateAppWin** for a more detailed description of these styles.

vVScrollBar A variable in which to return the handle of the window's vertical scroll-bar. If zero is returned, the window either does not have a vertical scroll-bar or its vertical scroll-bar is hidden.

See **CreateAppWin** for a more detailed description of App window scroll-bars.

See Also

AppWinSetMenuBar, **AppWinRemoveMenuBar**, **AppWinSetStyle**, **AppWinSetTitle**.

AppWinMaximize, AppWinMinimize

These subroutines allow the programmer to maximise and minimise an **AppWindow** contact.

- **AppWinMaximize** enlarges an App window to its maximum size, usually the size of the display.
- **AppWinMinimize** reduces an App window to an icon.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AppWinMaximize(*Context***,** *AppWindow***,** *vErr***)**

CALL AppWinMinimize(*Context***,** *AppWindow***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
AppWindow	The handle of the **AppWindow** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

An App window cannot be maximised unless it has a border style of **UIMS.BORDER** and a window style of **UIMS.SIZABLE**.

An App window cannot be minimised unless it has a border style of **UIMS.BORDER** and a window style of **UIMS.ICONISABLE**.

See Also

AppWinSetSizing, **AppWinRestore**.

AppWinRemoveMenuBar

This subroutine removes the **MenuBar** (if any) which is currently attached to an **AppWindow** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AppWinRemoveMenuBar(*Context***,** *AppWindow***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
AppWindow	The handle of the **AppWindow** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

AppWinSetMenuBar, **AppWinGetMenuBar**.

AppWinRestore

This subroutine restores a maximised or minimised **AppWindow** contact to its previous size.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AppWinRestore(*Context***,** *AppWindow***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
AppWindow	The handle of the **AppWindow** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

AppWinRestore has no effect if the App window is not maximised or minimised.

See Also

AppWinMinimize, **AppWinMaximize**, **AppWinSetSizing**.

AppWinSetDefButton – AppWinSetTitle

These subroutines change the different attributes of an **AppWindow** contact.

- **AppWinSetDefButton** sets which titled button within the window is the default.
- **AppWinSetMenuBar** attaches a menu bar to the App window.
- **AppWinSetSizing** sets whether the window is maximised, minimised or normal size.
- **AppWinSetStyle** changes the style of the window.
- **AppWinSetTitle** changes the title which appears at the top of the window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL AppWinSetDefButton(*Context***,** *AppWindow***,** *Button***,** *vErr***)**

CALL AppWinSetMenuBar(*Context***,** *AppWindow***,** *MenuBar***,** *vErr***)**

CALL AppWinSetSizing(*Context***,** *AppWindow***,** *Sizing***,** *vErr***)**

CALL AppWinSetStyle(*Context***,** *AppWindow***,** *WinStyle***,** *vErr***)**

CALL AppWinSetTitle(*Context***,** *AppWindow***,** *Title***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
AppWindow	The handle of the **AppWindow** contact.
Button	The handle of the **TitledButton** contact that is to be the default.
MenuBar	The handle of the **MenuBar** contact to be attached to the window. The new menu bar will replace that which is currently attached (if any).
Sizing	The required sizing state for the window. This must be one of the following values:

UIMS.WS.MAX	Maximise.
UIMS.WS.MIN	Minimise.
UIMS.WS.NORMAL	Normal/restored.

WinStyle The style of the window. This must be a combination of the following values:

UIMS.WIN.CLOSABLE	The window can be closed by the user.
UIMS.WIN.DIALOG	Permits movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
UIMS.WIN.HSCROLL	The window has a horizontal scroll-bar.
UIMS.WIN.ICONISABLE	The window has a minimise box.
UIMS.WIN.MOVABLE	The window can be moved by the user.
UIMS.WIN.SIZABLE	The size of the window can be changed by the user.
UIMS.WIN.VSCROLL	The window has a vertical scroll-bar.

The following pre-defined styles are also available:

UIMS.WIN.ALL	The combination of all of the above.
UIMS.NONE	None of the above.

See **CreateAppWin** for a more detailed description of these styles and of App window scroll-bars.

Title The title to be displayed at the top of the window. Note that if the window has no title bar, the title will not be displayed.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

AppWinGetMenuBar, **AppWinRemoveMenuBar**, **CreateMenuBar**, **AppWinMaximize**, **AppWinMinimize**, **AppWinRestore**, **AppWinGetStyle**, **CreateAppWin**.

BitTest

This subroutine returns the state of a specified element in a composite value.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL BitTest(*Value*, *Bit*, *vState*)

Syntax Elements

Value	The value containing the element you wish to test.
Bit	The element you wish to test.
vState	A variable in which to return the state of the element. This will be 1 if the element concerned is selected or 0 if the element is not selected.

Comments

BitTest allows the programmer to determine the settings of individual elements in the composite values returned by certain UIMS subroutines.

Value will normally be a composite value returned by a UIMS subroutine.

Bit will normally be a value defined in UIMSDEFS.

Example

The following fragment of code determines whether or not a dialog box can be moved.

```
* First fetch the style of the dialog box
CALL DlgBoxGetStyle(DLGBOX❶, STYLE)

* Then pass the result to BitTest to find out if it is movable
CALL BitTest(STYLE, UIMS.WIN.MOVABLE❷, MOVABLE)
IF MOVABLE THEN PRINT "This dialog box can be moved."
```

In this example:

❶ DLGBOX is a variable containing the handle of the dialog box.

❷ UIMS.WIN.MOVABLE is a constant defined in UIMSDEFS.

BrushGetColour

This subroutine returns the foreground colour of a **Brush** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL BrushGetColour(*Context***,** *Brush***,** *vColour***)**

Syntax Elements

Context	The handle of the application context.
Brush	The handle of the **Brush** object.
vColour	A variable in which a value representing the colour of the brush will be returned. This value will be a UIMS logical colour or an RGB value (see Appendix B).
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

BrushSetColour.

BrushSetColour

This subroutine changes the colour of a **Brush** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL BrushSetColour(*Context***,** *Brush***,** *Colour***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Brush	The handle of the **Brush** object.
Colour	The colour of the brush. This must be a UIMS logical colour or an RGB value (see Appendix B). If zero is specified a default of black will be used.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

BrushGetColour.

ChangeNVButtonGroup

This subroutine changes the titles of the buttons in a NewView button group and the response strings generated by them. It can also be used to control whether or not buttons in the group are visible.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS
INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL ChangeNVButtonGroup(*Context***,** *Group***,** *Control***,** *aTitles***,** *aResponses***,** *vErr***)**

Syntax Elements

Context The handle of the application context.

Group The identifier of the required contact group.

Control Whether or not the mapped states of the buttons will be changed. This must be one of the following values:

NV.CHANGE.MAP	Change the mapped states of the buttons, as specified in the *aTitles* parameter.
UIMS.NONE	Change only the titles and responses.

aTitles A dynamic array, each attribute of which must contain a string to be displayed as the title of one of the buttons in the group. If any attribute contains a null string, the title of the corresponding button will remain unchanged.

If the *Control* parameter is set to **NV.CHANGE.MAP**, buttons for which there are attributes in this array will be mapped (made visible) and the remainder unmapped (hidden).

aResponses A dynamic array, each attribute of which must contain a string that will be returned to the application when a button in the group is operated. If any attribute contains a null string, the response generated by the corresponding button will remain unchanged.

Only the characters with the ASCII values X'08' to X'0D', and X'20' (space) to X'7E' (tilde) can be used in a response string. If other characters are required, they must be specified as follows:

CHAR(11): '*XX*'

where '*XX*' is a hexadecimal value made up of two ASCII characters in the range '0' to '9' and 'A' to 'F' (upper case only).

For example, the BEL character (ASCII 7) is specified as follows:

```
CHAR(11):'07'
```

Note that if the VT character (X'0B') is required, it must be specified as CHAR(11):'0B'.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Example

The NewView button group BGRP1 contains five buttons, all of which are currently displayed. Only three buttons are now required, and the titles and responses of these are to be changed.

```
EQUATE AM TO CHAR(254)
.
.
.
TITLES = "Main":...
         AM:...
         AM:"Back"
RESPONSES = "M":CHAR(13):...
            AM:"K":CHAR(13):...
            AM
CALL ChangeNVButtonGroup(CONTEXT, ...
                         BGRP1, ...
                         NV.CHANGE.MAP, ...
                         TITLES, ...
                         RESPONSES, ...
                         ERR)
```

When the above code has executed, the following changes will have been made:

- Only the first three buttons in the group will be displayed (the other two will still exist, but will be hidden).
- The first button will have the title "Main" and it will generate the response string "M", followed by a carriage return.

- The second button's title will be unchanged, but it will now generate the response string "K", followed by a carriage return.
- The third button's response string will be unchanged, but its title will now be "Back".

See Also

ChangeNVButtonGroup, **ChangeNVContacts**, **CreateNVContactGroup**, **DestroyNVGroup**.

ChangeNVContacts

This subroutine changes the response strings generated by contacts in a NewView group.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS
INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL ChangeNVContacts(*Context***,** *Group***,** *FirstContact***,** *Number***,** *aResponses***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Group	The identifier of the required contact group.
FirstContact	The handle of the first contact in the group to be changed.
Number	The number of contacts to be changed.
aResponses	A dynamic array, each attribute of which must contain a string that will be returned to the application when a contact in the group is operated. The number of attributes in the array must be the same as the *number* parameter.

Only the characters with the ASCII values X'08' to X'0D', and X'20' (space) to X'7E' (tilde) can be used in a response string. If other characters are required, they must be specified as follows:

CHAR(11): '*XX*'

where '*XX*' is a hexadecimal value made up of two ASCII characters in the range '0' to '9' and 'A' to 'F' (upper case only).

For example, the BEL character (ASCII 7) is specified as follows:

```
CHAR(11):'07'
```

Note that if the VT character (X'0B') is required, it must be specified as CHAR(11):'0B'.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreateNVContactGroup, **DestroyNVGroup**.

CheckButtonDeselect

This subroutine deselects the specified **CheckButton** contact, clearing the 'X' (if any) displayed in its check box.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CheckButtonDeselect(*Context***,** *Button***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **CheckButton** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CheckButtonSelect, **CheckButtonSetSelected**, **CheckButtonGetSelected**.

CheckButtonGetSelected

This subroutine returns the current state (selected or deselected) of a **CheckButton** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CheckButtonGetSelected(*Context***,** *Button***,** *vSelected***)**

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **CheckButton** contact.
vSelected	A variable in which to return the state (selected or deselected) of the button. This will be one of the following values:

TRUE	The button is selected.
FALSE	The button is not selected.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CheckButtonSetSelected, **CheckButtonSetTitle**, **CheckButtonSetToggle**.

CheckButtonSelect

This subroutine selects the specified **CheckButton** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CheckButtonSelect(*Context***,** *Button***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **CheckButton** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

When a check button is selected an 'X' is displayed in its check box.

See Also

CheckButtonDeselect, **CheckButtonSetSelected**, **CheckButtonGetSelected**.

CheckButtonSetSelected – CheckButtonSetToggle

These subroutines change the attributes of a specified **CheckButton** contact.

- **CheckButtonSetSelected** sets the button to selected or deselected.
- **CheckButtonSetTitle** changes the title displayed beside the button.
- **CheckButtonSetToggle** changes the auto-toggle state of the button.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CheckButtonSetSelected(*Context***,** *Button***,** *Selected***,** *vErr***)**

CALL CheckButtonSetTitle(*Context***,** *Button***,** *Title***,** *vErr***)**

CALL CheckButtonSetToggle(*Context***,** *Button***,** *Toggle***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **CheckButton** contact.
Selected	The required button state. This must be one of the following values:

TRUE	Select the button.
FALSE	Deselect the button.

Title	The new title for the button.
Toggle	The required auto-toggle state. This must be one of the following values:

TRUE	Enable auto-toggle.
FALSE	Disable auto-toggle.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments When a check button is selected an 'X' is displayed in its check box.

See Also **CheckButtonGetSelected**.

ChildWinGetHScroll – ChildWinGetVScroll

These subroutines return the different attributes of a **ChildWindow** contact.

- **ChildWinGetHScroll** returns the handle of the Child window's horizontal scroll-bar, if any.
- **ChildWinGetStyle** returns the style of the Child window.
- **ChildWinGetVScroll** returns the handle of the Child window's vertical scroll-bar, if any.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ChildWinGetHScroll(*Context***,** *ChildWindow***,** *vHScrollBar***)**

CALL ChildWinGetStyle(*Context***,** *ChildWindow***,** *vWinStyle***)**

CALL ChildWinGetVScroll(*Context***,** *ChildWindow***,** *vVScrollBar***)**

Syntax Elements

Context The handle of the application context.

ChildWindow The handle of the **ChildWindow** contact.

vHScrollBar A variable in which to return the handle of the window's horizontal scroll-bar. If zero is returned, the window either does not have a horizontal scroll-bar or its horizontal scroll-bar is hidden.

See **CreateChildWin** for a more detailed description of Child window scroll-bars.

vWinStyle A variable in which a value representing the style of the window will be returned. This value will be a combination of one or more of the following:

UIMS.WIN.DIALOG	Permits movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
UIMS.WIN.HSCROLL	The window has a horizontal scroll-bar.
UIMS.WIN.TEXT	The window has a text canvas attached.
UIMS.WIN.VSCROLL	The window has a vertical scroll-bar.

The **BitTest** subroutine can be used to test the individual elements which make up the returned value.

See **CreateChildWin** for a more detailed description of these styles.

vVScrollBar	A variable in which to return the handle of the window's vertical scroll-bar. If zero is returned, the window either does not have a vertical scroll-bar or its vertical scroll-bar is hidden. See **CreateChildWin** for a more detailed description of Child window scroll-bars.

See Also

ChildWinSetStyle.

ChildWinSetDefButton, ChildWinSetStyle

These subroutines change the different attributes of an **ChildWindow** contact.

- **ChildWinSetDefButton** sets which titled button within the window is the default.
- **ChildWinSetStyle** changes the style of the Child window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ChildWinSetDefButton(*Context***,** *ChildWindow***,** *Button***,** *vErr***)**

CALL ChildWinSetStyle(*Context***,** *ChildWindow***,** *WinStyle***,** *vErr***)**

Syntax Elements

Context The handle of the application context.

ChildWindow The handle of the **ChildWindow**.

Button The handle of the **TitledButton** contact that is to be the default.

WinStyle The style of the window. This must be a combination of the following values:

UIMS.WIN.DIALOG	Permits movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
UIMS.WIN.HSCROLL	The window has a horizontal scroll-bar.
UIMS.WIN.VSCROLL	The window has a vertical scroll-bar.

The following pre-defined style is also available:

UIMS.NONE	None of the above.

See **CreateChildWin** for a more detailed description of these styles and of Child window scroll-bars.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

ChildWinGetStyle.

ClipboardGetContent, ClipboardGetSize

These subroutines provide access to the clipboard.

- **ClipboardGetContent** returns the contents of the clipboard.
- **ClipboardGetSize** returns the amount of data on the clipboard.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ClipboardGetContent(*DataFormat***,** *vContent***,** *vLength***)**

CALL ClipboardGetSize(*Format***,** *vSize***)**

Syntax Elements

DataFormat	The format in which to return the data from the clipboard. This must be a string up to four characters long. The following are recognised formats: "TEXT" ASCII text. "PICT" Reserved for future use. Other, application defined, formats can also be used.
vContent	A variable in which to return the data from the clipboard.
vLength	A variable in which to return the number of bytes of data returned in *vContent*.
Format	The format (see above) for which the size of the data is required.
vSize	A variable in which to return the length of the clipboard data. If the data on the clipboard is not in the requested format, zero is returned.

Comments

An alternative method of retrieving the clipboard contents is with the **Paste** subroutine.

See Also

Copy, **Cut**, **ClipboardSetContent**, **Paste**.

ClipboardSetContent

This subroutine places data on the clipboard.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ClipboardSetContent(*Format***,** *Content***,** *Length***,** *vErr***)**

Syntax Elements

Format	The format of the data to be placed on the clipboard. The following text strings are recognised formats: "TEXT" ASCII text. "PICT" Reserved for future use. Other, application defined, text strings can also be used.
Content	The data to place on the clipboard.
Length	The length of the data to be placed on the clipboard.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Alternative methods of placing data on the clipboard are the **Copy** and **Cut** subroutines.

See Also

Copy, **Cut**, **Paste**, **ClipboardGetContent**, **ClipboardGetSize**.

Copy

This subroutine is used to place on the clipboard, part or all of the data from an **EditBox** or **TextEditor** contact. The contents of the contact remain unchanged.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Copy(*Context***,** *Contact***,** *StartChar***,** *StartLine***,** *EndChar***,** *EndLine***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
StartChar	The character position of the start of the copy. The position must be specified as the number of characters from the start of the line specified in *StartLine*.
StartLine	The number of the line containing the position of the start of the copy. If *Contact* is the handle of an **EditBox**, this parameter will be ignored.
EndChar	The character position of the end of the copy. The position must be specified as the number of characters from the start of the line specified in *EndLine*.
EndLine	The number of the line containing the position of the end of the copy. If *Contact* is the handle of an **EditBox**, this parameter will be ignored.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

If *StartChar*, *StartLine*, *EndChar* and *EndLine* are all zero, all the data in the contact will be copied to the clipboard.

If *StartChar*, *StartLine*, *EndChar* and *EndLine* are all -1, the currently selected data will be copied to the clipboard.

If *Contact* is handle of a contact other than an **EditBox** or **TextEditor**, an error will be returned.

See Also

Cut, **Paste**, **ClipboardSetContent**, **ClipboardGetContent**, **ClipboardGetState**.

CreateAppWin

This subroutine creates an **AppWindow** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateAppWin(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *BorderStyle***,** *Parent***,** *vAppWindow***)**

Syntax Elements

Context	The handle of the context to which the App window will belong.
Ident	An integer value to use as the handle for the **AppWindow** contact. If this parameter is zero a handle will be assigned by UIMS and returned in the *vAppWindow* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title to be displayed at the top of the window. Note that if the window has no title bar, the title will not be displayed.
HPos	The horizontal position of the window in coordinate units. This specifies the position of the left-hand edge of the window, relative to the left-hand edge of the screen.
VPos	The vertical position of the window in coordinate units. This specifies the position of the top edge of the window, relative to the top edge of the screen.
Width	The overall width of the window in coordinate units.
Height	The overall height of the window in coordinate units.
Style	The style of the window. This must be a combination of the following values:

UIMS.WIN.CLOSABLE	The window can be closed by the user.
UIMS.WIN.DIALOG	Permits movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
UIMS.WIN.HSCROLL	The window has a horizontal scroll-bar.

UIMS.WIN.ICONISABLE	The window has a minimise box.
UIMS.WIN.MOVABLE	The window can be moved by the user.
UIMS.WIN.SIZABLE	The size of the window can be changed by the user.
UIMS.WIN.TEXT	The window has a text canvas attached.
UIMS.WIN.VSCROLL	The window has a vertical scroll-bar.

The following pre-defined styles are also available:

UIMS.WIN.ALL	The combination of all of the above, except **UIMS.WIN.TEXT**.
UIMS.NONE	None of the above.

BorderStyle The style of the window's border. This must be one of the following values:

UIMS.BORDER	Give the window a border.
UIMS.NONE	No border.

Parent The handle of the parent of the window, if required. The parent must be the application context. If a parent is specified, the window will be drawn immediately.

If *Parent* is a null string, the window is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vAppWindow A variable in which to return the handle of the newly-created App window. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

Window Styles The different window and border styles have the following effects:

UIMS.WIN.CLOSABLE
Generates a single border (overriding the border style), a title bar, and a system menu with the Close and Move commands enabled.

UIMS.WIN.ICONISABLE
Generates a single border (overriding the border style), a title bar, a minimise box, and a system menu with the Move and Minimize commands enabled.

UIMS.WIN.MOVABLE
Generates a single border (overriding the border style), a title bar, and a system menu with the Move command enabled.

UIMS.WIN.SIZABLE
Generates a double border (overriding the border style and any other window styles), a title bar, a maximise box, and a system menu with the Size, Maximize and Move commands enabled.

UIMS.WIN.HSCROLL and **UIMS.WIN.VSCROLL**
Generate a single border (overriding the border style) and the appropriate scroll-bar.

UIMS.WIN.TEXT
This creates a text canvas on which text strings and their positions in the client area are stored.

Notes:

1. A window with a system menu, title bar and border can always be moved by the user, whether or not style **UIMS.WIN.MOVABLE** is selected.

2. If a window is to have no border, or a border but no title bar, its style cannot include style elements **UIMS.WIN.MOVABLE**, **UIMS.WIN.CLOSABLE**, **UIMS.WIN.ICONISABLE** or **UIMS.WIN.SIZABLE**.

3. If the window does not have a title bar, the title of the window is not displayed.

Window Size and Position

HPos, *VPos*, *Width* and *Height* will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The position of the window is specified in screen-relative coordinates (position 0,0 is the top left-hand corner of the screen).

See Also

AppWinSetMenuBar, **AppWinSetSizing**, **AppWinSetStyle**, **AppWinSetTitle**, **CreateChildWin**.

CreateCheckButton

This subroutine creates a **CheckButton** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateCheckButton(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Parent***,** *vButton***)**

Syntax Elements

Context	The handle of the context to which the check button will belong.
Ident	An integer value to use as the handle for the **CheckButton** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vButton* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title to be displayed next to the check button.
HPos	The horizontal position of the button in coordinate units, relative to the left-hand edge of its parent's client area.
VPos	The vertical position of the button in coordinate units, relative to the top edge of its parent's client area.
Width	The width of the button in coordinate units. This specifies the total width of the button graphics and the title. If *Width* is specified as zero, a button will be created just wide enough to contain the graphic and the title.
Height	The height of the button in coordinate units. If *Height* is specified as zero, a button will be created just tall enough to contain the graphic or the title, whichever is the taller.
Parent	The handle of the parent of the titled button, if required. This can be any type of window. If the parent is currently displayed the button will be drawn immediately. If *Parent* is a null string, the button is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vButton	A variable in which to return the handle of the newly-created button. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

HPos and *VPos* specify the position of the top left-hand corner of the button, relative to the top left-hand corner of its parent's client area (position 0,0).

See Also

CheckButtonSetSelected, **CheckButtonSetTitle**, **CheckButtonSetToggle**, **CreateOptionButton**, **CreateTitledButton**.

CreateChildWin

This subroutine creates a **ChildWindow** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateChildWin(*Context***,** *Ident***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *BorderStyle***,** *Parent***,** *vChildWindow***)**

Syntax Elements

Context	The handle of the context to which the Child window will belong.
Ident	An integer value to use as the handle for the **ChildWindow** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vChildWindow* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
HPos	The horizontal position of the window in coordinate units. This specifies the position of the left-hand edge of the window, relative to the left-hand edge of its parent's client area.
VPos	The vertical position of the window in coordinate units. This specifies the position of the top edge of the window, relative to the top edge of its parent's client area.
Width	The overall width of the window in coordinate units.
Height	The overall height of the window in coordinate units.
Style	The style of the window. This must be a combination of the following values:

UIMS.WIN.DIALOG	Permits movement from child to child with the TAB and SHIFT+TAB keys, as in a dialog box.
UIMS.WIN.HSCROLL	The window has a horizontal scroll-bar.
UIMS.WIN.TEXT	The window has a text canvas attached.
UIMS.WIN.VSCROLL	The window has a vertical scroll-bar.

The following pre-defined style is also available:

UIMS.NONE	None of the above.

BorderStyle The style of the window's border. This must be one of the following values:

UIMS.BORDER	Give the window a border.
UIMS.NONE	No border.

Parent The handle of the parent of the window, if required. This must be an **AppWindow**, a **ChildWindow**, a **DialogBox** or an **InclusiveGroup**. If the parent is currently displayed the window will be drawn immediately.

If *Parent* is a null string, the window is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vChildWindow A variable in which to return the handle of the newly-created Child window. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

HPos, *VPos*, *Width* and *Height* will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The position of the window is specified in screen-relative coordinates (position 0,0 is the top left-hand corner of the screen).

See Also

ChildWinSetStyle, **CreateAppWin**.

CreateDlgBox

This subroutine creates a **DialogBox** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateDlgBox(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *Parent***,** *vDlgBox***)**

Syntax Elements

Context	The handle of the context to which the dialog box will belong.
Ident	An integer value to use as the handle for the **DialogBox** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vDlgBox* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title to be displayed at the top of the dialog box. Note that if the dialog box has no title bar, the title will not be displayed.
HPos	The horizontal position in coordinate units of the left-hand edge of the dialog box, relative to the left-hand edge of its parent's client area.
VPos	The vertical position in coordinate units of the top edge of the dialog box, relative to the top edge of its parent's client area.
Width	The overall width of the dialog box in coordinate units.
Height	The overall height of the dialog box in coordinate units.
Style	The required style for the dialog box. This must be a combination of the following values:

UIMS.WIN.CLOSABLE	The dialog box can be closed by the user.
UIMS.WIN.MOVABLE	The dialog box can be moved by the user.

The following pre-defined styles are also available:

UIMS.NONE	No system menu or title bar; not movable or closable.
UIMS.DEFAULT	The default setting (movable and closable).

Parent The handle of the parent of the dialog box, if required. This can be the application context or an **AppWindow**. If the parent is currently displayed the dialog box will be drawn immediately.

If *Parent* is a null string, the dialog box is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vDlgBox A variable in which to return the handle of the newly-created dialog box. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

Size and Position *HPos*, *VPos*, *Width* and *Height* will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The position of the window is specified in parent-relative coordinates (position 0,0 is the top left-hand corner of the parent's client area). Note, however, that if the parent of the dialog box is the application context, the position must be specified relative to the top left-hand corner of the screen.

Mode When first created, a dialog box is application modal. This can be changed with **DlgBoxSetMode** if required.

See Also

DlgBoxSetDefButton, **DlgBoxSetMode**, **DlgBoxSetStyle**, **DlgBoxSetTitle**, **CreateMessageBox**.

CreateDrawBrush

This subroutine creates a **Brush** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateDrawBrush(*Context***,** *Ident***,** *Colour***,** *Style***,** *vBrush***)**

Syntax Elements

Context The handle of the context to which the brush object will belong.

Ident An integer value to use as the handle for the **Brush** object. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vBrush* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

Colour The colour of the brush. This must be a UIMS logical colour or an RGB value (see Appendix B). If zero is specified a default of black will be used.

Style The style of the brush. This must be one of the following values:

UIMS.BRUSH.SOLID	Solid colour.
UIMS.BRUSH.HOLLOW	Transparent.

vBrush A variable in which to return the handle of the newly-created **Brush** object. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

BrushSetColour, **CreateDrawPen**.

CreateDrawFont

This subroutine creates a **Font** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateDrawFont(*Context***,** *Ident***,** *Style***,** *TypeFace***,** *PointSize***,** *vFont***)**

Syntax Elements

Context The handle of the context to which the font will belong.

Ident An integer value to use as the handle for the **Font** object. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vFont* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

Style The style of the font. This must be a combination of the following:

UIMS.FONT.BOLD
UIMS.FONT.ITALIC
UIMS.FONT.OUTLINE
UIMS.FONT.UNDERLINE
UIMS.FONT.STRIKEOUT

If none of the above are required, the style should be set to **UIMS.NONE**.

In some typefaces not all the above are available. If a style that is not available is selected, UIMS will use the nearest equivalent.

TypeFace The handle of a **TypeFace** object. If this parameter is zero, the default typeface is used.

PointSize The required point size for the font.

The point size should one of those which is available for the selected typeface **-** use **TypeFaceGetPointSizes** to find out which sizes are available. If the requested size is not available, UIMS will try to create it by scaling one of the available sizes; if this cannot easily be done, the nearest equivalent will be selected. Note that some typefaces can be scaled to any size.

If this parameter is zero, the first size in the typeface's list is used.

vFont	A variable in which to return the handle of the newly-created **Font** object. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

FontSetPointSize, **TypeFaceGetPointSize**, **TypeFaceGetPointSizes**, **FontSetStyle**, **FontSetTypeFace**, **GetTypeFace**, **GetTypeFaces**.

CreateDrawPen

This subroutine creates a **Pen** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateDrawPen(*Context***,** *Ident***,** *Colour***,** *Width***,** *Style***,** *vPen***)**

Syntax Elements

Context	The handle of the context to which the pen will belong.
Ident	An integer value to use as the handle for the **Pen** object. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vPen* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Colour	The colour of the pen. This must be a UIMS logical colour or an RGB value (see Appendix B).
Width	The width, in pixels, of lines drawn by the pen. If the width is set to zero, the pen will draw the thinnest and/or most efficient lines available on the display platform.
Style	The style of the pen. This must be one of the following values: **UIMS.PEN.SOLID** A continuous line. **UIMS.PEN.HOLLOW** An invisible line. If this parameter is zero, the style is set to **UIMS.PEN.SOLID**.
vPen	A variable in which to return the handle of the newly-created **Pen** object. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

PenSetColour, **PenSetWidth**, **CreateDrawBrush**.

CreateDrawrule

This subroutine creates a **Drawrule** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateDrawrule(*Context***,** *Ident***,** *Foreground***,** *Background***,** *DrawMode***,** *TextMode***,** *vDrawrule***)**

Syntax Elements

Context The handle of the context to which the drawrule will belong.

Ident An integer value to use as the handle for the **Drawrule** object. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vDrawrule* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

Foreground The foreground colour for text output. This must be a UIMS logical colour or an RGB value (see Appendix B).

If this parameter is set to **UIMS.DEFAULT**, the foreground colour is set to that of the default **Drawrule**.

Background The background colour for text and graphics output. This must be a UIMS logical colour or an RGB value (see Appendix B).

If this parameter is set to **UIMS.DEFAULT**, the background colour is set to that of the default **Drawrule**.

DrawMode The drawing mode used for graphics (pen and brush) output. This must be one of the following values:

UIMS.DRAW.CLEAR
UIMS.DRAW.COPY
UIMS.DRAW.NOTCLEAR
UIMS.DRAW.NOTCOPY
UIMS.DRAW.NOTOR
UIMS.DRAW.NOTXOR

UIMS.DRAW.OR
UIMS.DRAW.XOR

If this parameter is zero, the drawing mode will be set to **UIMS.DRAW.COPY**.

The effects of the different graphics drawing modes are described in Appendix B.

TextMode The drawing mode used for text output. This must be one of the following values:

UIMS.TEXT.OPAQUE	Fill the text background with the selected background colour.
UIMS.TEXT.HOLLOW	Do not fill the text background.

If this parameter is zero, text mode will be set to **UIMS.TEXT.OPAQUE**.

vDrawrule A variable in which to return the handle of the newly-created **Drawrule** object. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The default **Brush**, **Font** and **Pen** objects for the application context will be attached to the newly-created drawrule. These can be changed with the appropriate subroutines.

See Also

DrawruleSetBrush, **DrawruleSetColour**, **DrawruleSetFont**, **DrawruleSetPen**.

CreateEditBox

This subroutine creates an **EditBox** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateEditBox(*Context***,** *Ident***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *Mask***,** *Parent***,** *vEditBox***)**

Syntax Elements

Context	The handle of the application context to which the edit box will belong.
Ident	An integer value to use as the handle for the **EditBox** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vEditBox* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
HPos	The horizontal position in coordinate units of the left-hand edge of the edit box, relative to the left-hand edge of its parent's client area.
VPos	The vertical position in coordinate units of the top of the edit box, relative to the top edge of its parent's client area.
Width	The width of the edit box in coordinate units.
Height	The height of the edit box in coordinate units.
Style	The required style for the edit box. This must be one of the following values: **UIMS.EBOX.BORDER** Enclose the edit field in a box. **UIMS.NONE** Do not enclose the edit field in a box.
Mask	This parameter is for future use. It must be set to a string when calling **CreateEditBox**, but its value will be ignored.
Parent	The handle of the parent of the edit box, if required. This can be any type of window or an inclusive group. If the parent is currently displayed the edit box will be drawn immediately.

If *Parent* is a null string, the edit box is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vEditBox	A variable in which to return the handle of the newly-created edit box. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The position of the edit box is specified in parent-relative coordinates (position 0,0 is the top left-hand corner of the parent's client area).

The **EditBox** contact allows only a single line of text to be edited. To edit text with more than one line, use the **TextEditor** contact.

See Also

EditBoxSetContent, **EditBoxSetSelected**, **CreateTextEditor**.

CreateExGroup

This subroutine creates an **ExclusiveGroup** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateExGroup(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *Parent***,** *vGroup***)**

Syntax Elements

Context	The handle of the application context to which the exclusive group will belong.
Ident	An integer value to use as the handle for the **ExclusiveGroup** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vGroup* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title of the exclusive group.
HPos	The horizontal position in coordinate units of the left-hand edge of the group, relative to the left-hand edge of its parent's client area.
VPos	The vertical position in coordinate units of the top of the group, relative to the top edge of its parent's client area. Note that the top of the group is aligned with the top of the title text, not with the top of the bounding box.
Width	The width of the group in coordinate units.
Height	The height of the group in coordinate units. Note that this value must allow for the group title, which extends above the bounding box.
Style	The required style for the group. This can be either of the following values:

UIMS.BORDER	Enclose the group in a box.
UIMS.NONE	Do not enclose the group in a box.

Parent	The handle of the parent of the exclusive group, if required. This can be any type of window. If the parent is currently displayed the group will be drawn immediately. If *Parent* is a null string, the group is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.
vGroup	A variable in which to return the handle of the newly-created exclusive group. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The position of the group is specified in parent-relative coordinates (position 0,0 is the top left-hand corner of the parent's client area).

If the group has no bounding box, the title will not be displayed.

The children of an exclusive group must be **OptionButton** contacts. If any are not, the contact will not be created and zero will be returned.

See Also

CreateOptionButton, **CreateIncGroup**.

CreateIncGroup

This subroutine creates an **InclusiveGroup** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateIncGroup(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *Parent***,** *vGroup***)**

Syntax Elements

Context	The handle of the application context to which the inclusive group will belong.
Ident	An integer value to use as the handle for the **InclusiveGroup** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vGroup* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title of the inclusive group.
HPos	The horizontal position in coordinate units of the left-hand edge of the group, relative to the left-hand edge of its parent's client area.
VPos	The vertical position in coordinate units of the top of the group, relative to the top edge of its parent's client area. Note that the top of the group is aligned with the top of the title text, not with the top of the bounding box.
Width	The width of the group in coordinate units.
Height	The height of the group in coordinate units. Note that this value must allow for the group title, which extends above the bounding box.
Style	The required style for the group. This can be either of the following values:

UIMS.BORDER	Enclose the group in a box.
UIMS.NONE	Do not enclose the group in a box.

Parent	The handle of the parent of the inclusive group, if required. This can be any type of window. If the parent is currently displayed the group will be drawn immediately. If *Parent* is a null string, the group is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.
vGroup	A variable in which to return the handle of the newly-created inclusive group. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The position of the window is specified in parent-relative coordinates (position 0,0 is the top left-hand corner of the parent's client area).

If the group has no bounding box, the title will not be displayed.

Only the following types of contact can be attached as children of an inclusive group:

CheckButton	**ListBox**
ChildWindow	**OptionButton**
EditBox	**Rectangle**
ExclusiveGroup	**ScrollBar**
InclusiveGroup	**Text**
Line	**TextEditor**

See Also

CreateExGroup, **IncGroupSetStyle**, **IncGroupSetTitle**.

CreateLine

This subroutine creates a **Line** contact. The line is drawn between two specified points on the client area of the parent window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateLine(*Context***,** *Ident***,** *HStart***,** *VStart***,** *HEnd***,** *VEnd***,** *EndStyles***,** *Parent***,** *vLine***)**

Syntax Elements

Context The handle of the application context to which the **Line** contact will belong.

Ident An integer value to use as the handle for the **Line** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vLine* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

HStart The horizontal position in coordinate units of the start of the line.

VStart The vertical position in coordinate units of the start of the line.

HEnd The horizontal position in coordinate units of the end of the line, relative to the start of the line.

VEnd The vertical position in coordinate units of the end of the line, relative to the start of the line.

EndStyles This parameter is for future use. It must be set to a numeric value when calling **CreateLine**, but its value will be ignored.

Parent The handle of the parent of the line contact, if required. This can be any type of window. If the parent is currently displayed the line will be drawn immediately.

If *Parent* is a null string, the line is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vLine A variable in which to return the handle of the newly-created **Line** contact. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The position of the start of the line is specified in parent-relative coordinates (position 0,0 is the top left-hand corner of the parent's client area), using the coordinate mode (text or graphics) currently selected for the application context.

Other line attributes (width, colour, etc.) are set by means of a **Drawrule** object attached to the **Line** contact. Initially the drawing rule is that attached to the parent object, but this can be changed by calling the **SetDrawrule** subroutine.

See Also

SetDrawrule, **CreateRect**, **CreateText**.

CreateListBox

This subroutine creates a **ListBox** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateListBox(*Context***,** *Ident***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Controls***,** *Parent***,** *vListBox***)**

Syntax Elements

Context	The handle of the application context to which the list box will belong.
Ident	An integer value to use as the handle for the **ListBox** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vListBox* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
HPos	The horizontal position of the list box in coordinate units, relative to the left-hand edge of its parent's client area (position 0).
VPos	The vertical position of the list box in coordinate units, relative to the top edge of its parent's client area (position 0).
Width	The width of the list box in coordinate units.
Height	The height of the list box in coordinate units.
Controls	The required control settings for the list box. This can be either of the following values: **UIMS.LBOX.MULTISELECT** — Multiple selections allowed. **UIMS.NONE** — Allow only one item to be selected at a time.
Parent	The handle of the parent of the list box, if required. This can be any type of window. If the parent is currently displayed the list box will be drawn immediately.

If *Parent* is a null string, the list box is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vListBox A variable in which to return the handle of the newly-created list box. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

See Also

ListBoxAddContent, **ListBoxAddContents**, **ListBoxAddSelection**, **ListBoxAddSelections**, **ListBoxSetLink**.

CreateMenuBar

This subroutine creates a **MenuBar** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateMenuBar(*Context***,** *Ident***,** *Parent***,** *vMenuBar***)**

Syntax Elements

Context	The handle of the application context to which the menu bar will belong.
Ident	An integer value to use as the handle for the **MenuBar** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vMenuBar* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Parent	The handle of the parent of the **MenuBar** if required. If specified, this must be an **AppWindow**. If the parent is currently displayed the menu bar will be drawn immediately. If *Parent* is a null string, the contact is created without a parent and can be attached at a later time using **AppWinSetMenuBar**.
vMenuBar	A variable in which to return the handle of the newly-created **MenuBar**. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

CreatePullDownMenu, **MakePullDownMenu**, **CreateMenuItem**, **AppWinSetMenuBar**.

CreateMenuItem

This subroutine creates a **MenuItem** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateMenuItem(*Context***,** *Ident***,** *Title***,** *Parent***,** *vMenuItem***)**

Syntax Elements

Context	The handle of the application context to which the menu item will belong.
Ident	An integer value to use as the handle for the **MenuItem** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vMenuItem* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title of the menu item. An ampersand (&) preceding a character in this string designates that character as the selector key for the menu item. If a single hyphen is used as the title, a separator item is created. This appears as a continuous line across the width of its parent menu. A separator item cannot be selected by the user and should be used to visually group related menu items. Note that a separator item cannot be attached to a menu bar.
Parent	The handle of the parent of the menu item, if required. If specified, this must be either a **Menu** or a **MenuBar**. If the parent is currently displayed the menu item will be drawn immediately. If *Parent* is a null string, the contact is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.
vMenuItem	A variable in which to return the handle of the newly-created **MenuItem**. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

CreateMenuBar, **CreatePullDownMenu**, **MakePullDownMenu**, **AddChild**, **AddChildren**.

CreateMessageBox

This subroutine creates and displays a **MessageBox**.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateMessageBox(*Context***,** *Style***,** *Title***,** *Message***,** *aButtonTitles***,** *vResponse***,** *vErr***)**

Syntax Elements

Context The handle of the context to which the message box is to belong.

Style The style of the message box; that is, the number of buttons, the type of icon and which button is to be the default. This must be a combination of values, formed by adding together one from each of the following groups:

Number of Buttons	
0	Use pre-defined style.
1	One button.
2	Two buttons.
3	Three buttons.
Icon	
0	Use pre-defined styles.
16	Information icon.
32	Warning icon.
48	Alert icon.
64	Query icon.
Default button	
0	The left-most button is the default.
256	The second button is the default.
512	The third button is the default.

If the number of buttons is zero, the Icon value selects a pre-defined style, as follows:

Icon value	
16	Information icon and single OK button
32	Warning icon; OK and Cancel buttons
48	Alert icon; Retry and Cancel buttons
64	Query icon; OK and Cancel buttons

If no icon is specified, a pre-defined style is used. The following styles are available:

Pre-defined styles	
UIMS.INFO	Information icon and single OK button
UIMS.WARN2	Warning icon; OK and Cancel buttons
UIMS.WARN3	Warning icon; Yes, No and Cancel buttons
UIMS.ALERT2	Alert icon; Retry and Cancel buttons
UIMS.ALERT3	Alert icon; Abort, Retry and Ignore buttons
UIMS.QUERY2	Query icon; OK and Cancel buttons
UIMS.QUERY3	Query icon; Yes, No and Cancel buttons

Examples:

```
STYLE = 2 + 48
```

specifies two buttons and an Alert Icon. The first button is the default.

```
STYLE = UIMS.WARN3 + 256
```

specifies a Warning icon, and Yes, No and Cancel buttons. The No button is the default.

Title The title to be displayed at the top of the message box.

Message The message to be displayed. A newline character – CHAR(10) – can be used to start new a line where required.

aButtonTitles A dynamic array containing a list of button names (one in each attribute). If any attribute is a null string, a default button name will be used for the corresponding button (see *Style* parameter).

Note: If you are using a pre-defined style, this parameter should normally be a null string.

vResponse A variable in which to return a value representing the button that has been operated. The value will be one of the following:

Return value	Button operated
0	Leftmost button.
1	Next button.
2	Next button.
-1	ESC key

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

In addition to the message, the message box will contain a graphic icon appropriate to the type of message box specified.

A message box is always application modal.

See Also

CreateDlgBox.

CreateNVContactGroup

This subroutine creates a NewView contact group.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS
INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL CreateNVContactGroup(*Context***,** *Group***,** *FirstContact***,** *Number***,** *aResponses***,** *vErr***)**

Syntax Elements

Context The handle of the application context.

Group A unique user-assigned integer which will subsequently be used to identify the contact group.

FirstContact The handle of the first contact in the group.

Number The number of contacts in the group.

aResponses A dynamic array, each attribute of which must contain a string that will be returned to the application when a contact in the group is operated. The number of attributes in the array must be the same as the *Number* parameter.

Only the characters with the ASCII values X'08' to X'0D', and X'20' (space) to X'7E' (tilde) can be used in a response string. If other characters are required, they must be specified as follows:

CHAR(11): '*XX*'

where '*XX*' is a hexadecimal value made up of two ASCII characters in the range '0' to '9' and 'A' to 'F' (upper case only).

For example, the BEL character (ASCII 7) is specified as follows:

```
CHAR(11):'07'
```

Note that if the VT character (X'0B') is required, it must be specified as CHAR(11):'0B'.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The contact making up the group must have been previously created, or loaded using **LoadAppRes**.

The handles of the contacts in the group must be consecutive.

Only the following types of contact can be used in a NewView contact group:

MenuItem
TitledButton
CheckButton
OptionButton

Note that **CheckButton** and **OptionButton** contacts must have auto-toggling enabled. The required initial states of these types of button should be set before using the group.

See Also

ChangeNVButtonGroup, **ChangeNVContacts**, **SetEnabledNVGroup**, **SetMappedNVGroup**, **DestroyNVGroup**, **CheckButtonSetToggle**, **OptionButtonSetToggle**, **CheckButtonSetSelected**, **OptionButtonSetSelected**.

CreateNVHotspotGroup

This subroutine creates a group of NewView hot-spots within the application's terminal emulation window.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL CreateNVHotspotGroup(*Context***,** *Group***,** *Number***,** *aHPos***,** *aVPos***,** *aWidth***,** *aHeight***,** *aResponses***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Group	A unique user-assigned integer which will subsequently be used to identify the hot-spot group.
Number	The number of hot-spots in the group.
aHPos	A dynamic array, each attribute of which contains the horizontal position in text coordinates of one of the hot-spots in the group. Each position is relative to the left-hand edge of the TE window's terminal area.
aVPos	A dynamic array, each attribute of which contains the vertical position in text coordinates of one of the hot-spots in the group. Each position is relative to the top edge of the TE window's terminal area.
aWidth	A dynamic array, each attribute of which contains the width in text coordinates of one of the hot-spots in the group.
aHeight	A dynamic array, each attribute of which contains the height in text coordinates of one of the hot-spots in the group.
aResponses	A dynamic array, each attribute of which must contain a string that will be returned to the application when a hot-spot in the group is clicked with the mouse. Only the characters with the ASCII values X'08' to X'0D', and X'20' (space) to X'7E' (tilde) can be used in a response string. If other characters are required, they must be specified as follows: CHAR(11): '*XX*'

where '*XX*' is a hexadecimal value made up of two ASCII characters in the range '0' to '9' and 'A' to 'F' (upper case only).

For example, the BEL character (ASCII 7) is specified as follows:

```
CHAR(11):'07'
```

Note that if the VT character (X'0B') is required, it must be specified as CHAR(11):'0B'.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The TE window's terminal area is the current display page – that is, the area which receives all terminal output generated by the host. The size of the terminal area is defined in the RealLink Terminal Preferences and is unaffected by changes in the size of the TE window. *HPos* and *VPos* specify the positions of the top left-hand corners of the hot-spots, relative to the top left-hand corner (position 0,0) of this terminal area.

The *aHPos*, *aVPos*, *aWidth*, *aHeight* and *aResponses* arrays must contain exactly the same number of attributes as there are hot-spots.

See Also

DestroyNVGroup, **SetTeWindow**.

CreateOptionButton

This subroutine creates an **OptionButton** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateOptionButton(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Parent***,** *vButton***)**

Syntax Elements

Context	This is the handle of the context that the option button will belong to.
Ident	An integer value to use as the handle for the **OptionButton** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vButton* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title to be displayed next to the **OptionButton** graphics.
HPos	The horizontal position in coordinate units of the left-hand edge of the button, relative to the left-hand edge of its parent's client area (position 0).
VPos	The vertical position in coordinate units of the top edge of the button, relative to the top edge of its parent's client area (position 0).
Width	The width of the button in coordinate units. This specifies the total width of the button graphics and the title. If *Width* is specified as zero, a button will be created just wide enough to contain the graphic and the title.
Height	The height of the button in coordinate units. If *Height* is specified as zero, a button will be created just tall enough to contain the graphic or the title, whichever is the taller.
Parent	The handle of the parent of the option button. This may be any one of the window types. If the parent is currently displayed the button will be drawn immediately.
vButton	A variable in which to return the handle of the newly-created button. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in

the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

See Also

OptionButtonSetSelected, **OptionButtonSetTitle**, **OptionButtonSetToggle**, **CreateCheckButton**, **CreateTitledButton**.

CreatePointer

This subroutine creates a mouse **Pointer** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreatePointer(*Context***,** *Ident***,** *Type***,** *vPointer***)**

Syntax Elements

Context — The handle of the context to which the pointer will belong.

Ident — An integer value to use as the handle for the **Pointer** object. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vPointer* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

Type — The shape of the pointer. This must be one of the following values:

UIMS.PTR.ARROW	Standard arrow pointer.
UIMS.PTR.IBEAM	Text I-beam pointer.
UIMS.PTR.CROSS	Diagonal cross-hair pointer.
UIMS.PTR.PLUS	Horizontal and vertical cross-hair pointer.
UIMS.PTR.WAIT	Wait pointer **-** normally an hourglass.

vPointer — A variable in which to return the handle of the newly-created **Pointer** object. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

PointerSetType.

CreatePullDownMenu

This subroutine creates a **Menu** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreatePullDownMenu(*Context***,** *Ident***,** *Title***,** *Parent***,** *vMenu***)**

Syntax Elements

Context	The handle of the application context to which the menu will belong.
Ident	An integer value to use as the handle for the **Menu** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vMenu* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title of the menu. An ampersand (&) preceding a character in this string designates that character as the selector key for the menu.
Parent	The handle of the parent of the menu, if required. If specified, this must be either a **MenuBar** or another **Menu**. If the parent is currently displayed the menu will be drawn immediately. If *Parent* is a null string, the contact is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.
vMenu	A variable in which to return the handle of the newly-created **Menu**. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

See Also

MakePullDownMenu, **CreateMenuBar**, **CreateMenuItem**, **AddChild**, **AddChildren**.

CreateRect

This subroutine creates a **Rectangle** contact. The rectangle is drawn at a specified position on the client area of the parent window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateRect(*Context***,** *Ident***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *Parent***,** *vRect***)**

Syntax Elements

Context The handle of the application context to which the rectangle contact will belong.

Ident An integer value to use as the handle for the **Rectangle** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vRect* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

HPos The position of the left-hand edge of the rectangle in coordinate units, relative to the left-hand edge of its parent's client area (position 0).

VPos The position of the top edge of the rectangle in coordinate units, relative to the top edge of its parent's client area. (position 0).

Width The width of the rectangle in coordinate units.

Height The height of the rectangle in coordinate units.

Style The required style for the rectangle. This must be one of the following values:

UIMS.RECT.BORDER	Draw a rectangle with square corners.
UIMS.NONE	No border.

Parent The handle of the parent of the rectangle contact, if required. This can be any type of window. If the parent is currently displayed the rectangle will be drawn immediately.

If *Parent* is a null string, the rectangle is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vRect A variable in which to return the handle of the newly-created **Rectangle** contact. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

Other attributes (line width, foreground and background colours, etc.) are set by means of a **Drawrule** object attached to the **Rectangle** contact. Initially the drawrule is that attached to the parent object, but this can be changed by calling the **SetDrawrule** subroutine.

See Also

SetDrawrule, **CreateLine**, **CreateText**.

CreateScrollBar

This subroutine creates a **ScrollBar** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateScrollBar(*Context***,** *Ident***,** *Type***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Parent***,** *vScrollBar***)**

Syntax Elements

Context	The handle of the application context to which the scroll-bar will belong.
Ident	An integer value to use as the handle for the **ScrollBar** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vScrollBar* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Type	The orientation of the scroll-bar. This must be one of the following values: **UIMS.SCROLLBAR.VERT** Vertical scroll-bar. **UIMS.SCROLLBAR.HORZ** Horizontal scroll-bar.
HPos	The horizontal position in coordinate units of the left-hand edge of the scroll-bar, relative to the left-hand edge of its parent's client area (position 0).
VPos	The vertical position in coordinate units of the top of the scroll-bar, relative to the top edge of its parent's client area (position 0).
Width	The width of the scroll-bar in coordinate units.
Height	The height of the scroll-bar in coordinate units.
Parent	The handle of the parent of the scroll-bar, if required. This can be any type of window. If the parent is currently displayed the scroll-bar will be drawn immediately. If *Parent* is a null string, the scroll-bar is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vScrollBar	A variable in which to return the handle of the newly-created scroll-bar. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

When the scroll-bar is created its range, thumb position, and line and page increments will all be set to zero. Also, tracking will be off. Each of these attributes must be set by calling the appropriate subroutine (see below).

See Also

ScrollBarSetInc, **ScrollBarSetRange**, **ScrollBarSetThumb**, **ScrollBarSetTracking**.

CreateText

This subroutine creates a **Text** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateText(*Context***,** *Ident***,** *String***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Parent***,** *vText***)**

Syntax Elements

Context	The handle of the application context to which the **Text** contact will belong.
Ident	An integer value to use as the handle for the **Text** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vText* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
String	The text string to be displayed.
HPos	The horizontal position of the text in coordinate units, relative to the left-hand edge of its parent's client area.
VPos	The vertical position of the text in coordinate units, relative to the top edge of its parent's client area.
Width	The width of the containing window in coordinate units. If *Width* is specified as zero, a window wide enough to fit all the text onto a single line will be created.
Height	The height of the containing window in coordinate units. If *Height* is specified as zero, the text will be divided into separate lines, each *Width* or under in length, and the Text contact will be made tall enough to display all of the text.
Parent	The handle of the parent of the text contact, if required. This can be any type of window or an inclusive group. If the parent is currently displayed the text will be drawn immediately.

If *Parent* is a null string, the text is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vText A variable in which to return the handle of the newly-created **Text** contact. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

HPos and *VPos* specify the position of the top left-hand corner of the text, relative to the top left-hand corner of its parent's client area (position 0,0).

The text is initially left aligned. This can be changed with the **TextSetJustification** subroutine.

The text style (font, etc.) is specified in the associated drawrule (initially that attached to the parent window). This can be changed by using **SetDrawrule**.

Automatic Sizing

If *Width* and/or *Height* are specified as zero, the metrics of the font must be known in order to calculate the size of the contact. The contact's size is therefore set when it is attached to its parent. If its parent does not have a drawrule, the size is not set until its parent is itself given a parent. Refer also to the description of the Drawrule object in Chapter 3.

If both *Width* and *Height* are specified as zero, a window will be created large enough to fit all the text onto a single line.

The size of a Text contact can be recalculated by making it an orphan, setting its width and/or height to zero and then reattaching it to its parent.

See Also

DrawTextString, **TextSetContent**, **TextSetJustification**, **SetDrawrule**.

CreateTextEditor

This subroutine creates a **TextEditor** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateTextEditor(*Context***,** *Ident***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Style***,** *Parent***,** *vEditor***)**

Syntax Elements

Context The handle of the application context to which the text editor will belong.

Ident An integer value to use as the handle for the **TextEditor** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vEditor* parameter.

UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

HPos The horizontal position in coordinate units of the left-hand edge of the text editor, relative to the left-hand edge of its parent's client area.

VPos The vertical position in coordinate units of the top of the text editor, relative to the top edge of its parent's client area.

Width The width of the text editor in coordinate units.

Height The height of the text editor in coordinate units.

Style The required style for the text editor. This must be a combination of the following values:

UIMS.TXED.AUTOSCROLL	Autoscroll when the mouse is dragged outside the text editor window.
UIMS.TXED.BORDER	Enclose the text editor in a box.
UIMS.TXED.HSCROLLBAR	Provide a horizontal scroll-bar.
UIMS.TXED.READONLY	Display-only field; no editing allowed.
UIMS.TXED.VSCROLLBAR	Provide a vertical scroll-bar.

The value representing the required style is produced by adding the appropriate values together.

The following pre-defined styles are also available:

UIMS.DEFAULT	The default setting (all style components disabled).
UIMS.NONE	All style components disabled.

Parent The handle of the parent of the text editor, if required. This can be any type of window. If the parent is currently displayed the text editor will be drawn immediately.

If *Parent* is a null string, the text editor is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vEditor A variable in which to return the handle of the newly-created text editor. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

The top left-hand corner of the parent's client area is position 0,0.

See Also

TextEditorSetContent, **CreateEditBox**.

CreateTitledButton

This subroutine creates a **TitledButton** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL CreateTitledButton(*Context***,** *Ident***,** *Title***,** *HPos***,** *VPos***,** *Width***,** *Height***,** *Parent***,** *vButton***)**

Syntax Elements

Context	The handle of the application context to which the button will belong.
Ident	An integer value to use as the handle for the **TitledButton** contact. If this parameter is zero, a handle will be assigned by UIMS and returned in the *vButton* parameter. UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.
Title	The title to be displayed within the button, or the name of a file containing a bitmapped image.
HPos	The horizontal position of the button in coordinate units, relative to the left-hand edge of its parent's client area.
VPos	The vertical position of the button in coordinate units, relative to the top edge of its parent's client area.
Width	The width of the button in coordinate units. If *Width* is specified as zero, a button will be created just wide enough to contain the title or image.
Height	The height of the button in coordinate units. If *Height* is specified as zero, a button will be created just tall enough to contain the title or image.
Parent	The handle of the parent of the titled button, if required. This can be any type of window. If the parent is currently displayed the button will be drawn immediately. If *Parent* is a null string, the button is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.

vButton	A variable in which to return the handle of the newly-created button. If it could not be created for any reason, zero is returned. Note, however, that if asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Image files

Image files can be of the following types:

- Windows bitmaps (.BMP).
- Windows icon files (.ICO).
- Windows programs (.EXE).
- Windows dynamic link libraries (.DLL).

When specifying an image file, the full pathname should normally be given, including the drive letter and file-type extension. Note, however, that in the case of bitmap and icon files, the path can be omitted – the file will then be assumed to be in the directory specified in the Bitmaps entry in the [RealLink] section of the RFW.INI file on the PC.

Where a program, DLL or icon file contains more than one bitmap, the first will be displayed.

Comments

The *HPos*, *VPos*, *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

HPos and *VPos* specify the position of the top left-hand corner of the button, relative to the top left-hand corner of its parent's client area (position 0,0).

The following limitations apply to **TitledButton** contacts that contain images:

- **TitledButton** contacts containing images can only be created with the **CreateTitledButton** subroutine. It is not possible to specify an image in a resource script.
- The **TitledButtonSetStyle** and **TitledButtonSetTitle** subroutines cannot be used to change the appearance of a titled button that contains an image.
- **TitledButtonSetTitle** cannot be used to substitute an image for the title of an existing button.

See Also

TitledButtonSetStyle, **TitledButtonSetTitle**, **CreateCheckButton**, **CreateOptionButton**.

Cut

This subroutine is used to cut and place on the clipboard, part or all of the data from an **EditBox** or **TextEditor** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Cut(*Context***,** *Contact***,** *StartChar***,** *StartLine***,** *EndChar***,** *EndLine***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
StartChar	The character position of the start of the cut. The position must be specified as the number of characters from the start of the line specified in *StartLine*.
StartLine	The number of the line containing the position of the start of the cut. If *Contact* is the handle of an **EditBox**, this parameter will be ignored.
EndChar	The character position of the end of the cut. The position must be specified as the number of characters from the start of the line specified in *EndLine*.
EndLine	The number of the line containing the position of the end of the cut. If *Contact* is the handle of an **EditBox**, this parameter will be ignored.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

If *StartChar*, *StartLine*, *EndChar* and *EndLine* are all zero, all the data in the contact will be cut to the clipboard.

If *StartChar*, *StartLine*, *EndChar* and *EndLine* are all -1, the currently selected data will be cut to the clipboard.

If *Contact* is handle of a contact other than an **EditBox** or **TextEditor**, an error will be returned.

See Also

Copy, **ClipboardSetContent**, **Paste**, **ClipboardGetContent**, **ClipboardGetState**.

DDE.ADVISE

Obtains data from an 'advise' dynamic-data exchange (DDE) link established with **DDE.OPENADVISE**.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL DDE.ADVISE(*LinkIdent*, *vData*, *vStatus***)**

Syntax Elements

LinkIdent	A value, returned by the **DDE.OPENADVISE** subroutine, that identifies the required DDE conversation.
vData	A variable in which to return the contents of the conversation item.
vStatus	This is a variable that must be supplied to return the completion status of the subroutine. The value returned will be one of the following:

ADV.NODATA	The conversation item has not changed since **DDE.ADVISE** was last called. The contents of *vData* should be ignored.
ADV.MOREDATA	The conversation item has changed more than once since **DDE.ADVISE** was last called. *vData* contains the result of the first change. To obtain the result of the next change, **DDE.ADVISE** must be called again.
ADV.LASTDATA	The conversation item has changed once since **DDE.ADVISE** was last called. *vData* contains the result of this change.

Any other value indicates an error. Refer to Appendix D for a list of DDE error codes.

Comments

An 'advise' DDE conversation maintains a link to the application, topic and item specified in the call to **DDE.OPENADVISE**. Each time the item changes, the result is returned to UIMS, which adds it to a first-in-first-out buffer. **DDE.ADVISE** removes one item from this buffer and returns it to the calling application. The value returned in the *vStatus* parameter indicates whether the stack was empty, contained only a single item, or contains more data.

See Also

DDE.OPENADVISE, **DDE.CLOSEADVISE**, **DDE.PEEK**.

DDE.CLOSEADVISE

Closes an 'advise' dynamic-data exchange (DDE) link that was established with **DDE.OPENADVISE**.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL DDE.CLOSEADVISE(*LinkIdent*, *vErr***)**

Syntax Elements

LinkIdent	A value, returned by the **DDE.OPENADVISE** subroutine, that identifies the required DDE conversation
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion; if an error occurs, one of the DDE error codes listed in Appendix D is returned.

Note: **DDE.CLOSEADVISE** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

Comments

The server application is not closed by **DDE.CLOSEADVISE**.

See Also

DDE.OPENADVISE, **DDE.ADVISE**.

DDE.EXECUTE

This subroutine initiates a dynamic-data exchange (DDE) conversation with a Windows application and then sends the specified command or commands to that application. The application is started if it is not already running. On completion, the DDE conversation is terminated.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL DDE.EXECUTE(*Application***,** *Topic***,** *Command*, *vErr***)**

Syntax Elements

Application	The name used to specify a Windows application that supports DDE as a DDE server. This is usually the name of the application's .EXE file without the .EXE filename extension.
Topic	The name of a topic recognised by *Application*. An open document is a typical topic (if *Topic* is a document name, the document must be open). If *Application* does not recognise *Topic*, **DDE.EXECUTE** returns an error code.
Command	The command or commands to be executed by the server application.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion. If the DDE conversation could not be initiated, one of the DDE error codes listed in Appendix D is returned.

Note: **DDE.EXECUTE** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

Comments

Many applications that support DDE recognise a topic named System, which is always available and can be used to find out which other topics are available. For more information on the System topic, see **DDE.PEEK**.

If a **DDE.EXECUTE** command string contains an invalid command, an error will occur in the server application (if the application is minimised, its icon will flash). The **DDE.EXECUTE** function will not return until this error message has been acknowledged by the user.

If *Application* is started by **DDE.EXECUTE**, it continues running when the subroutine returns.

See Also **DDE.POKE**.

DDE.OPENADVISE

Establishes an 'advise' dynamic-data exchange (DDE) link to a Windows application. The application is started if it is not already running.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL DDE.OPENADVISE(*Application***,** *Topic***,** *Item*, *vLinkIdent*, *vErr***)**

Syntax Elements

Application	The name used to specify a Windows application that supports DDE as a DDE server. This is usually the name of the application's .EXE file without the .EXE filename extension.
Topic	The name of a topic recognised by *Application*. An open document is a typical topic. (If *Topic* is a document name, the document must be open.) If *Application* does not recognise *Topic*, **DDE.OPENADVISE** returns an error code.
Item	An item within a DDE topic recognised by the server application. **DDE.OPENADVISE** returns the entire contents of the specified item. If the server application does not recognise *Item*, an error code is returned.
vLinkIdent	A variable in which to return a value that identifies the DDE link. This identifier must be used when calling the **DDE.ADVISE** and **DDE.CLOSEADVISE** subroutines.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion. If the DDE conversation could not be initiated, one of the DDE error codes listed in Appendix D is returned.

Note: **DDE.OPENADVISE** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

Comments

The result of any change to the conversation item can be obtained by calling the **DDE.ADVISE** subroutine.

See Also

DDE.CLOSEADVISE, **DDE.ADVISE**.

DDE.PEEK

This subroutine initiates a dynamic-data exchange (DDE) conversation with a Windows application and then requests an item of information from that application. The application is started if it is not already running. On completion, the DDE conversation is terminated.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL DDE.PEEK(*Application***,** *Topic***,** *Item, vData, vErr***)**

Syntax Elements

Application	The name used to specify a Windows application that supports DDE as a DDE server. This is usually the name of the application's .EXE file without the .EXE filename extension.
Topic	The name of a topic recognised by *Application*. An open document is a typical topic. (If *Topic* is a document name, the document must be open.) If *Application* does not recognise *Topic*, **DDE.PEEK** returns an error code.
Item	An item within a DDE topic recognised by the server application. **DDE.PEEK** returns the entire contents of the specified item. If the server application does not recognise *Item*, an error code is returned.
vData	A variable in which to return the contents of the specified item.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion. If the DDE conversation could not be initiated, one of the DDE error codes listed in Appendix D is returned.

Note: **DDE.PEEK** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

The System Topic

Microsoft Excel and other applications that support DDE recognise a topic named System. The following lists three standard items in the System topic.

SysItems	Returns a list of all items in the System topic.
Topics	Returns a list of available topics.
Formats	Returns a list of all the supported Clipboard formats.

Note that you can get a list of the other items in the System topic by using the item SysItems.

Comments

If *Application* is started by **DDE.PEEK**, it continues running when the subroutine returns.

If the item is not recognised by the server, no data is returned.

See Also

DDE.ADVISE.

DDE.POKE

This subroutine initiates a dynamic-data exchange (DDE) conversation with a Windows application and then sends data to that application. The application is started if it is not already running. On completion, the DDE conversation is terminated.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL DDE.POKE(*Application***,** *Topic***,** *Item, Data, vErr***)**

Syntax Elements

Application	The name used to specify a Windows application that supports DDE as a DDE server. This is usually the name of the application's .EXE file without the .EXE filename extension.
Topic	The name of a topic recognised by *Application*. An open document is a typical topic. (If *Topic* is a document name, the document must be open.) If *Application* does not recognise *Topic*, **DDE.POKE** returns an error code.
Item	An item within a DDE topic recognised by the server application. If the server application does not recognise *Item*, an error code is returned.
Data	The data to send to the server application.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion. If the DDE conversation could not be initiated, one of the DDE error codes listed in Appendix D is returned.

Notes:

1. **DDE.POKE** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

2. If you specify a non-existent item in a call to **DDE.POKE**, no error is returned.

Comments

If *Application* is started by **DDE.POKE**, it continues running when the subroutine returns.

See Also

DDE.EXECUTE.

Destroy

This subroutine destroys an object or contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Destroy(*Context***,** *Object***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Object	The handle of the object or contact you wish to destroy.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Any children attached to the object will also be destroyed.

If you destroy an application's root window this will have the effect of making the application invisible.

DestroyNVGroup

This subroutine destroys a NewView group created with **CreateNVContactGroup** or **CreateNVHotspotGroup**.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMSDEFS FROM UIMS-TOOLS ;* Only required for contact groups.
INCLUDE UIMSCOMMON FROM UIMS-TOOLS ;* Only required for contact groups.

CALL DestroyNVGroup(*Context***,** *Group***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Group	The identifier for the group to be destroyed.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreateNVContactGroup, **CreateNVHotspotGroup**.

Disable

This subroutine disables a specified contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Disable(*Context***,** *Contact***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact you wish to disable.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A disabled contact remains displayed, but cannot be selected by the user. The disabled state is indicated by a greying effect, the exact form of which is platform dependent.

See Also

Destroy, **Enable**, **SetEnabled**, **GetState**.

DisplayGetMetrics, DisplayGetPixelSize

These subroutines return the different attributes of a **Display** object.

- **DisplayGetMetrics** returns information about the size of various window elements when shown on the specified display.
- **DisplayGetPixelSize** returns the size in pixels of the display image.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DisplayGetMetrics(*Context***,** *Display***,** *vBdrWidth***,** *vBdrHeight***,** *vSizeBdrWidth***,** *vSizeBdrHeight***,** *vTitleBarHeight***,** *vMenuBarHeight***,** *vVScrollWidth***,** *vHScrollHeight***,** *vErr***)**

CALL DisplayGetPixelSize(*Context***,** *Display***,** *vPWidth***,** *vPHeight***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Display	The handle of the **Display** object.
vBdrWidth	A variable in which to return the width in pixels of a non-sizeable (single-width) border.
vBdrHeight	A variable in which to return the height in pixels of a non-sizeable (single-width) border.
vSizeBdrWidth	A variable in which to return the width in pixels of a sizeable (double-width) border.
vSizeBdrHeight	A variable in which to return the height in pixels of a sizeable (double-width) border.
vTitleBarHeight	A variable in which to return the height in pixels of a title bar.
vMenuBarHeight	A variable in which to return the height in pixels of a menu bar.
vVScrollWidth	A variable in which to return the width in pixels of a vertical scroll-bar.
vHScrollHeight	A variable in which to return the height in pixels of a horizontal scroll-bar.

vPWidth	A variable in which to return the width in pixels of the display device.
vPHeight	A variable in which to return the height in pixels of the display device.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

While all the attributes of a **Display** object can be read with these subroutines, different platforms may offer differing capabilities. For those attributes that are not supported on a particular display platform, the subroutine concerned should return the appropriate error code (see Appendix D). Note, however, that this is not guaranteed, and that the values returned may be invalid.

See Also

AppWinGetDisplay, **GetDefaults**.

DisplayImage

This subroutine displays the contents of a specified image file in the RealLink window, or in an App or Child window. The image can be in any one of the following formats:

- Windows bitmap (.BMP).
- Windows Metafile (.WMF).
- Tagged Image File Format (.TIF).
- PC Paintbrush (.PCX).
- CompuServe GIF (.GIF).
- Truevision Targa (.TGA).

Syntax

INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DisplayImage(*ImageMan*, *Left*, *Top*, *Right*, *Bottom*, *ImageFile*, *ScaleFactor*, *Window*, *Context*, *vImage*)

Syntax Elements

ImageMan	The handle of the image manager, returned by the **StartImage** subroutine.
Left	The position of the left-hand edge of the image, relative to the left-hand edge of the containing window's client area.
Top	The position of the top edge of the image, relative to the top edge of the containing window's client area.
Right	The position of the right-hand edge of the image, relative to the left-hand edge of the containing window's client area.
Bottom	The position of the bottom edge of the image, relative to the top edge of the containing window's client area.
ImageFile	A string containing the path and name of the image file.
ScaleFactor	This parameter is for future use. A value must be supplied, but will be ignored.

Window	The handle of the window in which to display the image. If this parameter is zero, the image is displayed in the currently active 'terminal emulation' (TE) window.
Context	The handle of the application context. If the *Window* parameter is zero, this must also be set to zero.
vImage	A variable in which to return the handle of the displayed image. If, for any reason, it could not be created, zero is returned.

Comments

If *Window* is zero, the *HPos*, *VPos*, *Width* and *Height* parameters must be specified in text coordinates. Otherwise they must be in graphics coordinates.

The image is scaled to fit within the area defined by the *Left*, *Top*, *Right* and *Bottom* parameters. It is not possible to crop the image.

If the image file cannot be found, a message is displayed and *vImage* is returned set to 0. If you do not wish the user to see this message, you should use the **SystemCommand** subroutine to check that the image file exists before calling **DisplayImage**.

Dynamic Imaging Libraries

For each supported image format the RealLink directory contains a Dynamic Imaging Library (DIL) file. The names of the DIL files are constructed as follows:

IMG *format* **.DIL**

where *format* is, in most cases, the same as the file name extension of the image file to be displayed – for example, the PCX DIL is called IMGPCX.DIL. (At present, the only exception to this rule is the TIFF DIL, where the image file name extension is TIF, but the *format* part of the DIL file name is TIFF.)

If, when you call **DisplayImage** the image file name extension does not correspond to any of the available DILs, a message is displayed and the *vImage* parameter is returned set to 0. If you do not wish the user to see this message, you should use the **SystemCommand** subroutine to check that the appropriate DIL exists before calling **DisplayImage**.

See Also

EraseImage, **StartImage**, **StopImage**.

DlgBoxGetMode, DlgBoxGetStyle

These subroutines return the different attributes of a **DialogBox** contact.

- **DlgBoxGetMode** returns the mode of the dialog box.
- **DlgBoxGetStyle** returns the style of the dialog box.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DlgBoxGetMode(*Context***,** *DlgBox***,** *vMode***)**

CALL DlgBoxGetStyle(*Context***,** *DlgBox***,** *vStyle***)**

Syntax Elements

Context	The handle of the application context.
DlgBox	The handle of the **DialogBox** contact.
vMode	A variable in which to return the mode of the dialog box. The value returned will be one of the following:

UIMS.MODE.ALLAPPS	UIMS application modal.
UIMS.MODE.APP	Application modal.
UIMS.MODE.LESS	Modeless.
UIMS.MODE.SYS	System modal.

vStyle	A variable in which the style of the dialog box will be returned. The value returned is a combination of one or more of the following:

UIMS.WIN.CLOSABLE	The dialog box can be closed by the user.
UIMS.WIN.MOVABLE	The dialog box can be moved by the user.

The **BitTest** subroutine can be used to test the individual elements which make up the returned value.

See **CreateDlgBox** for a more detailed description of these styles.

See Also

DlgBoxSetMode, **DlgBoxSetStyle**.

DlgBoxSetDefButton – DlgBoxSetTitle

These subroutines change the different attributes of a **DialogBox** contact.

- **DlgBoxSetDefButton** sets which titled button within the dialog box is the default.
- **DlgBoxSetMode** sets the mode of the dialog box.
- **DlgBoxSetStyle** changes the style of the dialog box.
- **DlgBoxSetTitle** changes the title which appears at the top of the dialog box.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DlgBoxSetDefButton(*Context***,** *DlgBox***,** *Button***,** *vErr***)**

CALL DlgBoxSetMode(*Context***,** *DlgBox***,** *Mode***,** *vErr***)**

CALL DlgBoxSetStyle(*Context***,** *DlgBox***,** *Style***,** *vErr***)**

CALL DlgBoxSetTitle(*Context***,** *DlgBox***,** *Title***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
DlgBox	The handle of the **DialogBox** contact.
Button	The handle of the **TitledButton** contact that is to be the default.
Mode	The required mode for the dialog box. This must be one of the following values:

UIMS.MODE.ALLAPPS	UIMS application modal – applications launched from the current instance of RealLink cannot be used until the dialog box is cleared.
UIMS.MODE.APP	Application modal – the current UIMS application cannot be used until the dialog box is cleared.
UIMS.MODE.LESS	Modeless – does not prevent the use of the current or any other application.

UIMS.MODE.SYS	System modal – no application can be used until the dialog box is cleared.

When first created, a dialog box is application modal.

Style The required style for the dialog box. This must be a combination of the following values:

UIMS.WIN.CLOSABLE	The dialog box can be closed by the user.
UIMS.WIN.MOVABLE	The dialog box can be moved by the user.

The following pre-defined styles are also available:

UIMS.NONE	No system menu or title bar; not movable or closable.
UIMS.DEFAULT	The default setting (movable and closable).

See **CreateDlgBox** for a more detailed description of these styles.

Title The title to be displayed above the dialog box. Note that if the style of the dialog box is **UIMS.NONE**, the title will not be displayed.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreateDlgBox, **DlgBoxGetMode**, **DlgBoxGetStyle**.

Draw

This subroutine draws a contact on the display.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Draw(*Context***,** *Contact***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact you wish to draw.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Draw bypasses the current update mode and immediately draws the contact.

The contact must be mappable and have a mappable parent before it can be drawn. If the contact is an orphan, or it or its parent are unmappable, the draw operation will fail and an error code will be returned.

See Also

Move, **Destroy**, **Resize**.

DrawLine, DrawRect

These subroutines draw graphics elements on the client area of the specified window.

- **DrawLine** draws a line. If required, the line can have arrowheads at the ends.
- **DrawRect** draws a rectangle.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DrawLine(*Context***,** *Contact***,** *HStart***,** *VStart***,** *HEnd***,** *VEnd***,** *EndStyles***,** *vErr***)**

CALL DrawRect(*Context***,** *Contact***,** *Left***,** *Top***,** *Right***,** *Bottom***,** *RectStyle***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the window contact. This must be an App window, a Child window, a dialog box or an inclusive group.
HStart	The horizontal position in coordinate units of the start of the line.
VStart	The vertical position in coordinate units of the start of the line.
HEnd	The horizontal position in coordinate units of the end of the line.
VEnd	The vertical position in coordinate units of the end of the line.
EndStyles	This parameter is for future use. It must be set to a numeric value when calling **DrawLine**, but its value will be ignored.
Left	The position of the left-hand edge of the rectangle in coordinate units, relative to the left-hand edge of its parent's client area.
Top	The position of the top edge of the rectangle in coordinate units, relative to the top edge of its parent's client area.
Right	The position of the right-hand edge of the rectangle in coordinate units, relative to the left-hand edge of its parent's client area.

Bottom	The position of the bottom edge of the rectangle in coordinate units, relative to the top edge of its parent's client area.
RectStyle	The required style for the rectangle. This must be one of the following values:

UIMS.RECT.BORDER	Draw a rectangle with square corners.
UIMS.NONE	No border.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The positions of the start and end of the line, and the edges of the rectangle are specified in parent-relative coordinates (position 0,0 is the top left-hand corner of the parent's client area), using the coordinate mode (text or graphics) currently selected for the application context.

Other attributes (line width, colours, etc.) are determined by the **Drawrule** object attached to the window contact (see Chapter 3).

See Also

CreateLine, **CreateRect**, **DrawTextString**.

DrawruleGetBrush – DrawruleGetPen

These subroutines return the different attributes of a **Drawrule** object.

- **DrawruleGetBrush** returns the handle of the **Brush** object that is attached to the drawrule.
- **DrawruleGetColour** returns the foreground and background colours.
- **DrawruleGetFont** returns the handle of the **Font** object that is attached to the drawrule.
- **DrawruleGetPen** returns the handle of the **Pen** object that is attached to the drawrule.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DrawruleGetBrush(*Context***,** *Drawrule***,** *vBrush***)**

CALL DrawruleGetColour(*Context***,** *Drawrule***,** *vForeground***,** *vBackground***,** *vErr***)**

CALL DrawruleGetFont(*Context***,** *Drawrule***,** *vFont***)**

CALL DrawruleGetPen(*Context***,** *Drawrule***,** *vPen***)**

Syntax Elements

Context	The handle of the application context.
Drawrule	The handle of the **Drawrule** object.
vBrush	A variable in which to return the handle of the **Brush** object which is attached to the drawrule.
vForeground	A variable in which to return the foreground colour
vBackground	A variable in which to return the background colour
vFont	A variable in which to return the handle of the **Font** object which is attached to the drawrule.
vPen	A variable in which to return the handle of the **Pen** object which is attached to the drawrule.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

DrawruleSetBrush, **DrawruleSetColour**, **DrawruleSetFont**, **DrawruleSetPen**.

DrawruleSetBrush – DrawruleSetPen

These subroutines change the attributes of a specified **Drawrule** object.

- **DrawruleSetBrush** changes the **Brush** object attached to the drawrule.
- **DrawruleSetColour** changes the foreground and background colours.
- **DrawruleSetFont** changes the **Font** object attached to the drawrule.
- **DrawruleSetPen** changes the **Pen** object attached to the drawrule.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DrawruleSetBrush(*Context***,** *Drawrule***,** *Brush***,** *vErr***)**

CALL DrawruleSetColour(*Context***,** *Drawrule***,** *Foreground***,** *Background***,** *vErr***)**

CALL DrawruleSetFont(*Context***,** *Drawrule***,** *Font***,** *vErr***)**

CALL DrawruleSetPen(*Context***,** *Drawrule***,** *Pen***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Drawrule	The handle of the **Drawrule** object.
Brush	The handle of the **Brush** object to be attached to the drawrule. If this parameter is zero, the application context default brush is attached. The new brush replaces that previously attached
Foreground	The foreground colour for text output. This must be a UIMS logical colour or an RGB value (see Appendix B).
Background	The background colour for text and graphics output. This must be a UIMS logical colour or an RGB value (see Appendix B). If this parameter is zero, the background colour will be set to white.

Font	The handle of the **Font** object to be attached to the drawrule. If this parameter is zero, the application context default font is attached. The new font replaces that previously attached
Pen	The handle of the **Pen** object to be attached to the drawrule. If this parameter is zero, the application context default pen is attached. The new pen replaces that previously attached
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreateDrawrule, **DrawruleGetBrush**, **DrawruleGetColour**, **DrawruleGetFont**, **DrawruleGetPen**.

DrawTextString

This subroutine draws text on the client area or text canvas of the specified window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL DrawTextString(*Context***,** *Contact***,** *Text***,** *HPos***,** *VPos***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the window contact. This must be an App window, a Child window, a dialog box or an inclusive group.
Text	The text string to be drawn.
HPos	The horizontal position of the text in coordinate units, relative to the left-hand edge of its parent's client area.
VPos	The vertical position of the text in coordinate units, relative to the top edge of its parent's client area.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The *HPos* and *VPos* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

HPos and *VPos* specify the position of the top left-hand corner of the text, relative to the top left-hand corner of its parent's client area (position 0,0).

The text style (font, etc.) is determined by the **Drawrule** attached to the window contact.

See Also

CreateText, **DrawLine**, **DrawRect**.

EditBoxGetContent

This subroutine returns the text contents of an **EditBox** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL EditBoxGetContent(*Context***,** *EditBox***,** *vText***,** *vComplete***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
EditBox	The handle of the **EditBox** contact.
vText	A variable in which to return the text currently contained in the edit box.
vComplete	This parameter is for future use. A variable must be supplied, but it will always be returned set to zero.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

EditBoxSetContent.

EditBoxSetContent, EditBoxSetSelected

These subroutines change the different attributes of an **EditBox** contact.

- **EditBoxSetContent** assigns a text string to the edit box for editing or display.
- **EditBoxSetSelected** selects all or part of the text in the edit box.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL EditBoxSetContent(*Context***,** *EditBox***,** *Text***,** *vErr***)**

CALL EditBoxSetSelected(*Context***,** *EditBox***,** *StartPos***,** *EndPos***,** *State***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
EditBox	The handle of the **EditBox** contact
Text	The text string to be displayed for editing in the edit box window. The characters are entered as if typed at the keyboard.
StartPos	The position of the first selected character.
EndPos	The position of the last selected character.
State	Whether the text between *StartPos* and *EndPos* is to be selected or deselected. This must be one of the following values:
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

State values:

TRUE	Select the text.
FALSE	Deselect the text.

Comments

The first (left-most) character in the edit box is at position 0.

When calling **EditBoxSetSelected**, if both *StartPos* and *EndPos* are set to zero, the entire contents of the edit box will be selected.

See Also

CreateEditBox, **EditBoxGetContent**.

Enable

This subroutine enables a previously disabled contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Enable(*Context***,** *Contact***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact to be enabled.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A disabled contact is displayed on the screen, but cannot be selected by the user. The disabled state is indicated a greying effect, the exact form of which is platform dependent. This subroutine removes the greying effect and permits the user to select the contact.

See Also

Destroy, **Disable**, **SetEnabled**, **GetState**.

Erase

This subroutine erases a part of a contact's the client area, or the whole of the text canvas (if any). The erased area is filled with the current background colour, as specified by the **Drawrule** attached to the contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Erase(*Context***,** *Contact***,** *Left***,** *Top***,** *Right***,** *Bottom***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
Left	The position of the left-hand edge of the area to be erased.
Top	The position of the top edge of the area to be erased.
Right	The position of the right-hand edge of the area to be erased.
Bottom	The position of the bottom edge of the area to be erased.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Contact must be the handle of an **AppWindow**, **ChildWindow**, **DialogBox** or **InclusiveGroup**. If the handle of any other type of contact is specified, an error is returned.

If *Contact* specifies an App or Child window, **Erase** will clear the client area of the window.

If *Left*, *Right*, *Top* and *Bottom* are all zero, the whole of the client area will be erased.

If *Left*, *Right*, *Top* and *Bottom* are all set to -1, the text canvas (if any) and the whole of the client area will be erased.

See Also

DrawLine, **DrawRect**, **DrawTextString**.

EraseImage

This subroutine removes an image displayed in the current TE window, or in an App or Child window.

Syntax

INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL EraseImage(*ImageMan*, *Image*, *vErr*)

Syntax Elements

ImageMan	The handle of the image manager, returned by the **StartImage** subroutine.
Image	The handle of the displayed image, returned by the **DisplayImage** subroutine.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion. Otherwise, one of the error codes listed in Appendix D is returned.

Note: **EraseImage** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

Comments

If the image was displayed in the current TE window, this is redrawn. Otherwise the erased area is filled with the current background colour, as specified by the **Drawrule** attached to the window.

See Also

DisplayImage, **StartImage**, **StopImage**.

Execute

This subroutine starts a Windows application on the PC.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL Execute(*CommandLine***,** *WindowState***,** *Control***,** *vErr***)**

Syntax Elements

CommandLine A string containing the name of the program, plus any optional parameters and/or switches. If the program name does not contain a directory path, UIMS will search the PC for the executable file as follows:

1. The currently selected directory on the PC.
2. The Windows program directory (that containing WIN.COM).
3. The Windows system directory (that containing KERNEL.COM).
4. The directories listed in the PATH environment variable.

WindowState Specifies how the window containing the program should appear. This must be one of the following values:

EXECUTE.HIDE
Hides the window and passes activation to another window.

EXECUTE.MAXIMIZE
The same as **EXECUTE.SHOWMAXIMIZED**.

EXECUTE.MINIMIZE
Minimises the specified window and activates the top-level window in the window manager's list.

EXECUTE.NORMAL
The same as **EXECUTE.SHOWNORMAL**.

EXECUTE.RESTORE
The same as **EXECUTE.SHOWNORMAL**.

EXECUTE.SHOW
Activates a window and displays it in its current size and position.

EXECUTE.SHOWMAXIMIZED
Activates the window and displays it as a maximised window.

EXECUTE.SHOWMINIMIZED	Activates the window and displays it as an icon.
EXECUTE.SHOWMINNOACTIVE	Displays the window as an icon. The window that is currently active remains active.
EXECUTE.SHOWNA	Displays the window in its current state. The window that is currently active remains active.
EXECUTE.SHOWNOACTIVATE	Displays a window in its most recent size and position. The window that is currently active remains active.
EXECUTE.SHOWNORMAL	Activates and displays the window. If the window is minimised or maximised, UIMS restores it to its original size and position.

Control Specifies whether or not the subroutine should complete before returning to the calling application. This value will be a combination of one or more of the following:

EXECUTE.SINGLE	Do not start a second instance the program if it is already running.
EXECUTE.REFOCUS	Return the focus to the calling application once the called program is running.
EXECUTE.WAIT	Do not return until the called application has been closed.
RFW.NONE	Do not return to the calling application.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will be set to **ERR.RFW.SUCCESS** for successful completion or will contain one of the Execute error codes listed in Appendix D.

Comments

Execute cannot be used to start non-windows applications.

See Also

SystemCommand, **SendKeys**.

ExGroupGetSel

This subroutine returns the handle of the currently selected option button within an **ExclusiveGroup** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ExGroupGetSel(*Context***,** *Group***,** *vSelection***)**

Syntax Elements

Context	The handle of the application context.
Group	The handle of the **ExclusiveGroup** contact.
vSelection	A variable in which to return the handle of the currently selected option button.

See Also

OptionButtonGetSelected.

ExGroupSetStyle, ExGroupSetTitle

These subroutines change the different attributes of an **ExclusiveGroup** contact.

- **ExGroupSetStyle** changes the style of the group.
- **ExGroupSetTitle** changes the title displayed above the group.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ExGroupSetStyle(*Context***,** *Group***,** *Style***,** *vErr***)**

CALL ExGroupSetTitle(*Context***,** *Group***,** *Title***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Group	The handle of the **ExclusiveGroup** contact.
Style	The required style for the group. This can be either of the following values:
Title	The new group title.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Values for *Style*:

UIMS.BORDER	Enclose the group in a box.
UIMS.NONE	Do not enclose the group in a box.

Comments

If the group has no bounding box, the title will not be displayed.

FontGetMetrics – FontGetTypeFace

These subroutines return the different attributes of a **Font** object.

- **FontGetMetrics** returns the metrics (dimensions) of the font.
- **FontGetPointSize** returns the font's point size.
- **FontGetStyle** returns the style of the font.
- **FontGetTextLen** returns the length of a string as it appears on the screen.
- **FontGetTypeFace** returns the typeface being used.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL FontGetMetrics(*Context***,** *Font***,** *vHeight***,** *vAscent***,** *vDescent***,** *vLeading***,** *vLcWidth***,** *vUcWidth***,** *vMaxWidth***,** *vErr***)**

CALL FontGetPointSize(*Context***,** *Font***,** *vPointSize***)**

CALL FontGetStyle(*Context***,** *Font***,** *vStyle***)**

CALL FontGetTextLen(*Context***,** *Font***,** *String***,** *vLength***)**

CALL FontGetTypeFace(*Context***,** *Font***,** *vTypeFace***)**

Syntax Elements

Context	The handle of the application context.
Font	The handle of the **Font** object.
vHeight	A variable in which to return the font height. The returned value is the sum of the ascent and the descent.
vAscent	A variable in which to return the height of the tallest character above the baseline.
vDescent	A variable in which to return the height of the longest descender below the baseline.

vLeading	A variable in which to return the distance between adjacent lines of type; that is, the distance between the bottom of the longest descender and the top of the tallest character when printed on adjacent lines.
vLcWidth	A variable in which to return the average width of a lower case character.
vUcWidth	A variable in which to return the average width of an upper case character.
vMaxWidth	A variable in which to return the width of the widest character.
vPointSize	A variable in which to return the selected point size.
vStyle	A variable in which a value representing the selected font style will be returned. This value will be a combination of one or more of the following: **UIMS.FONT.BOLD** **UIMS.FONT.ITALIC** **UIMS.FONT.OUTLINE** **UIMS.FONT.UNDERLINE** **UIMS.FONT.STRIKEOUT** The **BitTest** subroutine can be used to test the individual elements which make up the returned value.
String	A text string.
vLength	A variable in which to return the length of *String*. The value returned is the length in pixels, when *String* printed in the specified font.
vTypeFace	A variable in which to return the handle of the selected **TypeFace** object.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The values returned in *vHeight*, *vAscent*, *vDescent*, *vLeading*, *vLcWidth*, *vUcWidth* and *vMaxWidth* are all in pixels.

Refer to Chapter 3 for more details of font metrics.

See Also

FontSetPointSize, **FontSetStyle**, **FontSetTypeFace**.

FontSetPointSize – FontSetTypeFace

These subroutines change the attributes of a specified **Font** object.

- **FontSetPointSize** sets the point size of the font.
- **FontSetStyle** changes the font style.
- **FontSetTypeFace** changes the typeface.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL FontSetPointSize(*Context***,** *Font***,** *PointSize***,** *vErr***)**

CALL FontSetStyle(*Context***,** *Font***,** *Style***,** *vErr***)**

CALL FontSetTypeFace(*Context***,** *Font***,** *TypeFace***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Font	The handle of the **Font** object.
PointSize	The required point size for the font. The point size should one of those which is available for the selected typeface **-** use **TypeFaceGetPointSizes** to find out which sizes are available. If a size that is not available is requested, the closest match will be selected. If this parameter is zero, the first size in the typeface's list is used.
Style	The style of the font. This must be a combination of the following: **UIMS.FONT.BOLD** **UIMS.FONT.ITALIC** **UIMS.FONT.OUTLINE** **UIMS.FONT.UNDERLINE** **UIMS.FONT.STRIKEOUT** If none of the above are required, the style should be set to **UIMS.NONE**.

In some typefaces not all the above are available. If a style that is not available is selected, UIMS will use the nearest equivalent.

TypeFace	The handle of a **TypeFace** object. If this parameter is zero, the default typeface is used.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

FontGetPointSize, **TypeFaceGetPointSize**, **TypeFaceGetPointSizes**, **FontGetStyle**, **FontGetTypeFace**, **CreateDrawFont**.

GetAppName

This subroutine returns the name of the application – that is, the name passed to the **SignOn** subroutine.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetAppName(*Context***,** v*AppName***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
vAppName	A variable in which to return the name of the application.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

SignOn.

GetBorderStyle

This subroutine returns the border style of an App or Child window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetBorderStyle(*Context***,** *Contact***,** *vStyle***)**

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the window.
vStyle	A variable in which to return a value representing the border style. This value will be one of the following:

UIMS.BORDER	The window has a border.
UIMS.NONE	The window does not have a border.

See Also

SetBorderStyle.

GetChild – GetChildFocus

These subroutines return information about the children of an object.

- **GetChild** returns the handle of the child at a specified position in the list.
- **GetChildCount** returns the number of children attached to an object.
- **GetChildren** returns the complete list of children.
- **GetChildFocus** identifies which child within a contact currently has the focus.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetChild(*Context***,** *Object***,** *Index***,** *vChild***)**

CALL GetChildCount(*Context***,** *Object***,** *vCount***)**

CALL GetChildren(*Context***,** *Object***,** *vaChildren***,** *vErr***)**

CALL GetChildFocus(*Context***,** *Contact***,** *vFocus***)**

Syntax Elements

Context	The handle of the application context.
Object	The handle of the parent object.
Index	The position in the list of the child whose handle you require. The list is numbered starting from 0.
vChild	A variable in which to return the handle of the child.
vCount	A variable in which to return the number of children the object has.
vaChildren	A variable in which to return the list of children. The list will be returned as a dynamic array, with one handle to each attribute.
Contact	The handle of the parent contact.

vFocus	A variable in which to return the handle of the child that currently has focus. If zero is returned, none of the specified contact's children has a the focus.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

GetChildFocus always returns the handle of a child of the specified contact. If the child has children of its own, the focus may in fact lie with one of these latter children.

See Also

AddChild, **AddChildren**, **RemoveChild**, **RemoveChildren**, **GetObjectParent**, **GetFrontWindow**, **SetContactFocus**.

GetClip

This subroutine returns the boundary of a window's clipping region.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetClip(*Context***,** *Window***,** *vTop***,** *vLeft***,** *vBottom***,** *vRight***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Window	The handle of the window.
vTop	A variable in which to return the position of the top edge of the window's clipping region.
vLeft	A variable in which to return the position of the left-hand edge of the window's clipping region.
vBottom	A variable in which to return the position of the bottom edge of the window's clipping region.
vRight	A variable in which to return the position of the right-hand edge of the window's clipping region.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The values returned in *vTop*, *vLeft*, *vBottom* and *vRight* will depend on the coordinate mode (text or graphics) currently selected for the application context. In all cases the values are relative to the top left-hand corner of the client area (position 0,0).

If *vTop*, *vLeft*, *vBottom* and *vRight* are all zero, no clipping region has been set.

See Also

SetClip.

GetCoordMode

This subroutine returns the coordinate mode by which screen positions are referenced.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetCoordMode(*Context***,** *vCoordMode***)**

Syntax Elements

Context	The handle of the **AppContext** object.
vCoordMode	A variable in which to return a value representing the coordinate mode. This value will be one of the following:

UIMS.COORD.TEXT	Screen positions are referenced to the nearest character position, where the size of a character is that of an upper case character in the default system typeface.
UIMS.COORD.GRAPHIC	Screen positions are referenced to the nearest pixel.

Comments

When an application signs on to UIMS, text mode is selected.

See Also

SetCoordMode.

GetCursorPosition, GetCursorState

These subroutines return the different attributes of the cursor within an **AppWindow** or **ChildWindow** contact.

- **GetCursorPosition** returns the position of the text cursor within the window.
- **GetCursorState** returns the type of text cursor that is currently selected and whether or not the cursor is visible.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetCursorPosition(*Context***,** *Window***,** *vHPos***,** *vVPos***,** *vErr***)**

CALL GetCursorState(*Context***,** *Window***,** *vVisible***,** *vCurType***,** *vErr***)**

Syntax Elements

Context	The handle of the application context.
Window	The handle of the **AppWindow** or **ChildWindow** contact.
vHPos	A variable in which to return the horizontal position of the cursor, relative to the left-hand edge of the client area.
vVPos	A variable in which to return the vertical position of the cursor, relative to the top edge of the client area.
vVisible	A variable in which to return whether or not the cursor is visible. This will be one of the following values:

TRUE	The cursor is visible.
FALSE	The cursor is invisible.

vCurType	A variable in which to return a value representing the type of cursor being used in the window. This value will be one of the following:

UIMS.BAR	Vertical bar.
UIMS.BLOCK	Block cursor.
UIMS.OUTLINE	Outline cursor.
UIMS.UNDERLINE	Underline cursor.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The values returned in *vHPos* and *vVPos* will depend on the coordinate mode (text or graphics) currently selected for the application context. In all cases the values are relative to the top left-hand corner of the client area (position 0,0).

See Also

SetCursorPosition, **SetCursorState**.

GetDefaults

This subroutine returns the handles of the default **Display**, **Printer** and **TypeFace** objects from the **SystemDictionary**.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetDefaults(*vDisplay***,** *vPrinter***,** *vTypeFace***,** *vErr***)**

Syntax Elements

vDisplay	A variable in which to return the handle of the default **Display** object.
vPrinter	A variable in which to return the handle of the default **Printer** object. **Note:** Printer display objects are not supported on this version of UIMS. This parameter is provided for use on later releases.
vTypeFace	A variable in which to return the handle of the default **TypeFace** object.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

GetDrawrule.

GetDrawrule

This subroutine returns the handle of the **Drawrule** object that is attached to an object or contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetDrawrule(*Context***,** *Object***,** *vDrawrule***)**

Syntax Elements

Context	The handle of the application context.
Object	The handle of an object or contact.
vDrawrule	A variable in which to return the handle of the **Drawrule** object.

Comments

A drawrule can be attached to only the following objects and contacts:

AppWindow
ChildWindow
Line
Rectangle
Text
AppContext

If an object or contact other than those listed above is specified, *vDrawrule* will be returned set to zero.

See Also

SetDrawrule.

GetErrorText

This subroutine returns a textual description of a specified UIMS error.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetErrorText(*Error***,** *vText***,** *vErr***)**

Syntax Elements

Error	An error code returned by a UIMS subroutine.
vText	A variable in which to return the textual description of the error.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

GetEventMask

This subroutine returns the event mask applied to a specified object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetEventMask(*Context***,** *Object***,** *vEventMask***)**

Syntax Elements

Context	The handle of the application context.
Object	The handle of an object.
vEventMask	A variable in which to return a value representing the event mask setting for the specified object. This value will be a combination of the event mask constants listed in Chapter 4. The **BitTest** subroutine can be used to test the individual elements which make up the returned value.

See Also

SetEventMask, **GetSecondaryEventMask**.

GetFrontWindow

This subroutine returns the handle of the top window of an application.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetFrontWindow(*Context***,** *vTopWindow***)**

Syntax Elements

Context	The handle of the **AppContext**.
vTopWindow	A variable in which to return the handle of the top window in the specified **AppContext**.

Comments

The top window is that **AppWindow** which either currently has the focus or which contains the contact which currently has the focus. If some other application has the focus, the top window is that which last had the focus.

See Also

GetRootWindow, **GetChildFocus**.

GetHelpFile – GetHelpKey

These subroutines return the settings of the application's **AppHelp** object.

- **GetHelpFile** returns the name of the application's help file.
- **GetHelpIndex** returns the help-id of the help file section that is associated with a specified contact.
- **GetHelpKey** returns the virtual code of the key currently assigned as the help accelerator.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetHelpFile(*Context***,** *vFilename***,** *vErr***)**

CALL GetHelpIndex(*Context***,** *Contact***,** *vSection***)**

CALL GetHelpKey(*Context***,** *vKey***)**

Syntax Elements

Context	The handle of the **AppContext**.
vFilename	A variable in which to return the name of the help file.
Contact	The handle of a contact.
vSection	A variable in which to return the help-id of the section of the help file that is associated with the specified contact.
vKey	A variable in which to return the virtual key code of the key that is assigned as the help accelerator.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

SetHelpFile, **SetHelpIndex**, **SetHelpKey**.

GetMsg

This subroutine retrieves the next available message from the message queue.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetMsg(*TimeOut*, *vContext*, *vWindow*, *vContact*, *vMsgType*, *vTimeStamp*, *vData1*, *vData2*, *vData3*, *vData4*)

Syntax Elements

TimeOut	The time, in tenths of a second, that **GetMsg** will wait for a message if the queue is empty. If this parameter is zero, **GetMsg** will not return until a message is available.
vContext	A variable in which to return the handle of the application context in which the event occurred.
vWindow	A variable in which to return the handle of the window in which the event occurred.
vContact	A variable in which to return the handle of the contact in which the event occurred.
vMsgType	A variable in which to return the type of message. This will be one of the message types listed in Chapter 4. If **GetMsg** has returned because no message is available (see *TimeOut* parameter above), *vMsgType* will be zero.
vTimeStamp	A variable in which to return a value representing the time that the event occurred. Note that not all messages return a time-stamp value (see Chapter 4 for details).
vData1, *vData2*, *vData3*, *vData4*	Variables in which to return message-specific data.

Comments

For some types of message, *vContext*, *vWindow* and/or *vContact* are returned set to zero - refer to Chapter 4 for details.

See Also

SetEventMask, **GetEventMask**.

GetObjectParent

This subroutine returns the handle of a object's parent.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetObjectParent(*Context***,** *Object***,** *vParent***)**

Syntax Elements

Context	The handle of the application context.
Object	The handle of the object whose parent you require.
vParent	A variable in which to return the handle of the parent. If the object has no parent, zero will be returned.

See Also

GetChild, **GetChildren**.

GetPointer

This subroutine returns the handle of the **Pointer** object that is attached to an object or contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetPointer(*Context*, *Object*, *vPointer*)

Syntax Elements

Context	The handle of the application context.
Object	The handle of an object or contact.
vPointer	A variable in which to return the handle of the **Pointer** object.

Comments

A pointer can be attached to only the following objects and contacts:

AppWindow
ChildWindow
AppContext

If an object or contact other than those listed above is specified, *vPointer* will be returned set to zero.

See Also

SetPointer.

GetPointerPos

This subroutine returns the position of the mouse pointer, relative to either the screen or a specified contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetPointerPos(*Context*, *Contact*, *vHPos*, *vVPos*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of a contact. If this parameter is zero the position is returned relative to the screen.
vHPos	A variable in which to return the horizontal coordinate of the pointer's position.
vVPos	A variable in which to return the vertical coordinate of the pointer's position.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

GetPointerPos returns the position of the pointer's hot-spot. If a contact is specified, the values returned specify the position relative to the top left-hand corner of the contact's client area (position 0,0); otherwise the position is relative to the top left-hand corner of the screen.

The values returned will depend on the coordinate mode (text or graphics) currently selected for the application context.

See Also

SetPointerPos.

GetPosition

This subroutine returns the position of a contact relative to its parent.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetPosition(*Context*, *Contact*, *vHPos*, *vVPos*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact whose position you require.
vHPos	A variable in which to return the horizontal coordinate of the position.
vVPos	A variable in which to return the vertical coordinate of the position.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The values returned specify the position of the top left-hand corner of the contact, relative to the top left-hand corner of its parent's client area (position 0,0). Note, however, that for **ExclusiveGroup** and **InclusiveGroup** contacts, the vertical position returned is that of the top of the title text (even if none is displayed), rather than the top of the enclosing box.

The values returned will depend on the coordinate mode (text or graphics) currently selected for the application context.

Note: Contacts that can be moved by the user can be positioned to the nearest pixel, whichever coordinate mode is selected. In text mode, therefore, the values returned by **GetPosition** are accurate only to the nearest character position.

See Also

Move.

GetRootWindow

This subroutine returns the handle of the root window of an application.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetRootWindow(*Context*, *vRootWindow*)

Syntax Elements

Context	The handle of the **AppContext**.
vRootWindow	A variable in which to return the handle of the application's root window.

Comments

The root window is the first **AppWindow** contact created by the application.

See Also

GetFrontWindow, **GetChildFocus**.

GetSecondaryEventMask

This subroutine returns the secondary event mask which has been set for an application.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetSecondaryEventMask(*Context*, *vEventMask*, *vUnmaskable*, *vAlert*, *vErr*)

Syntax Elements

Context	The handle of the application context.
vEventMask	A variable in which to return a value representing the secondary event mask setting for the application. This value will be a combination of the event mask constants listed in Chapter 4. The **BitTest** subroutine can be used to test the individual elements which make up the returned value.
vUnmaskable	A variable in which to return whether messages which cannot be masked are allowed to reach the application. This must be one of the following values:

TRUE	Non-maskable messages are allowed to reach the application.
FALSE	Non-maskable messages are not allowed to reach the application.

vAlert	This parameter is for future use. A variable must be supplied, but it will always be returned set to **FALSE**.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The secondary event mask is described in Chapter 4.

See Also

SetSecondaryEventMask, **GetEventMask**.

GetSize

This subroutine returns the size of a contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetSize(*Context*, *Contact*, *vWidth*, *vHeight*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact whose size you require.
vWidth	A variable in which to return the width of the contact.
vHeight	A variable in which to return the height of the contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

For **ExclusiveGroup** and **InclusiveGroup** contacts, the size returned includes the space occupied by the border and the title text (even if none is displayed).

If the contact is minimised both values returned will be zero.

The values returned will depend on the coordinate mode (text or graphics) currently selected for the application context.

Note: The user can resize a contact to the nearest pixel, whichever coordinate mode is selected. In text mode, therefore, the values returned by **GetSize** are accurate only to the nearest character position.

See Also

Resize.

GetSolidColour

This subroutine returns the solid colour which is the closest available to a specified red, green and blue colour combination.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetSolidColour(*Context*, *Colour*, *vSolidColour*)

Syntax Elements

Context	The handle of the application context.
Colour	A UIMS logical colour or an absolute colour, specified as a combination of red, green and blue.
vSolidColour	A variable in which to return the closest available solid colour. This will be an absolute colour - that is a red, green and blue colour combination.

Comments

This subroutine should be used to ensure that a solid colour is used as the background to text or other foreground detail.

UIMS screen colours are described in detail in Appendix B.

GetState

This subroutine returns the state of a contact - whether or not it is mappable and whether or not it is enabled.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetState(*Context*, *Contact*, *vState*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
vState	A variable in which to return the state of the contact. The value returned is a bit-significant value consisting of the following elements:

UIMS.ENABLED	If set, the contact is enabled; if not, it is disabled.
UIMS.MAPPABLE	If set, the contact is mappable; if not, it is unmappable.

The **BitTest** subroutine can be used to test the individual elements which make up this value.

See Also

SetMapped, **Map**, **UnMap**, **SetEnabled**, **Enable**, **Disable**.

GetTeFontSize, GetTeFontSizes

These subroutines return information about the fonts available for use in the RealLink or currently active 'terminal emulation' (TE) window.

- **GetTeFontSize** returns the currently selected font size.
- **GetTeFontSizes** returns a list of the available font sizes.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetTeFontSize(*vPointSize***,** *vWidth***,** *vHeight***)**

CALL GetTeFontSizes(*vaPointSizes***,** *vaWidths***,** *vaHeights***)**

Syntax Elements

vPointSize	A variable in which to return the currently selected point size.
vWidth	A variable in which to return the width in pixels of characters in the currently selected point size.
vHeight	A variable in which to return the height in pixels of characters in the currently selected point size.
vaPointSizes	A variable in which to return a list of numbers representing the available point sizes. The list is returned as a dynamic array with one point size in each attribute.
vaWidths	A variable in which to return a list of numbers representing the widths in pixels of characters in the available point sizes. The list is returned as a dynamic array with one value in each attribute.
vaHeights	A variable in which to return a list of numbers representing the heights in pixels of characters in the available point sizes. The list is returned as a dynamic array with one value in each attribute.

Comments

The positions of the values returned in the *vaWidths* and *vaHeights* arrays correspond to the positions of the point sizes returned in the *vaPointSizes* parameter.

See Also

SetTeFontSize, **SetTeWindow**.

GetTypeFace, GetTypeFaces

These subroutines return the handles of **TypeFace** objects.

- **GetTypeFace** returns the handle of a specified typeface.
- **GetTypeFaces** returns a list of the typefaces available on the PC.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetTypeFace(*Index*, *vTypeFace*)

CALL GetTypeFaces(*vaTypeFaces*, *vErr*)

Syntax Elements

Index	The position in the list of the typeface whose handle you require. The list is numbered starting from 0.
vTypeFace	A variable in which to return the handle of the typeface.
vaTypeFaces	A variable in which to return the list of typefaces. The list will be returned as a dynamic array, with one handle to each attribute.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

It is not necessary to fetch the list of typefaces before calling **GetTypeFace**.

See Also

GetDefaults.

GetUimsVersion

This subroutine returns the UIMS version number and revision level.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetUimsVersion(*vMajor*, *vMinor*, *vRevision*, *vErr*)

Syntax Elements

vMajor	A variable in which to return the UIMS version number.
vMinor	A variable in which to return the UIMS release number.
vRevision	A variable in which to return the UIMS revision level.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The version number of a UIMS release is made up as follows:

UIMS *Major*.*Minor* Revision *Revision*

For example: UIMS 1.0 Revision D.

GetUpdate

This subroutine returns the update mode of a contact; that is, when the contact will be redrawn if a change occurs.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GetUpdate(*Context*, *Contact*, *vUpdate*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact whose update mode you require.
vUpdate	A variable in which to return the update mode. The value returned will be one of the following:

UIMS.IMMEDIATE	Redraw immediately.
UIMS.NONE	Don't redraw; wait for a **Draw** command.

See Also

SetUpdate.

GrabPointer

This subroutine causes mouse messages to be sent to a specified contact, regardless of the position of the pointer.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL GrabPointer(*Context*, *Contact*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact to which messages will be sent.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

In normal operation, pointer messages are sent to the contact in which the pointer is positioned. The position of the pointer is reported relative to the current contact. If the mouse is not within a contact, no pointer messages are generated.

GrabPointer causes all pointer messages to be diverted to a specified contact and the position of the pointer to be reported relative to that contact. In addition, pointer motion messages are generated periodically, even if the pointer does not move.

When the contact no longer requires all pointer messages, the application should call the **UngrabPointer** subroutine so that other contacts can receive pointer messages.

If pointer drag messages are enabled in a contact's event mask, when a drag event starts within that contact, UIMS will automatically perform a **GrabPointer**, followed by an **UngrabPointer** when the drag ends.

GrabPointer does not affect the movement of the pointer around the screen.

See Also

UngrabPointer.

HiByte

This subroutine returns the value of the most-significant byte of a word (2 byte) value.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL HiByte(*Word*, *vHiByte*)

Syntax Elements

Word	The word value from which you require the most-significant byte.
vHiByte	A variable in which to return the value of the most-significant byte.

Comments

HiByte allows the programmer to determine the values of the individual bytes in the composite word values returned by the **GetMsg** subroutine.

Word will normally be a composite value returned by **GetMsg**.

Example

The following fragment of code determines whether any other mouse buttons were held down when a mouse button was clicked.

```
* Wait for the next message
CALL GetMsg(0, ...
               CONTEXT, ...
               WINDOW, ...
               CONTACT, ...
               MSGTYPE, ...
               TIMESTAMP, ...
               DATA1, ...
               DATA2, ...
               DATA3, ...
               DATA4, ...
               ERR)

BEGIN CASE
CASE MSGTYPE = UIMS.MSG.CLICK
```

```
* Use HiByte to separate out the mouse buttons that are held down
    CALL HiByte(DATA1, MOUSE.MOD)
    IF MOUSE.MOD THEN
        PRINT "Another mouse button was held down."
    END ELSE
        PRINT "No other mouse buttons held down."
    END

END CASE
```

See Also

GetMsg, **LoByte**.

IncGroupSetStyle, IncGroupSetTitle

These subroutines change the different attributes of an **InclusiveGroup** contact.

- **IncGroupSetStyle** changes the style of the group.
- **IncGroupSetTitle** changes the title displayed above the group.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL IncGroupSetStyle(*Context*, *Group*, *Style*, *vErr*)

CALL IncGroupSetTitle(*Context*, *Group*, *Title*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Group	The handle of the **InclusiveGroup** contact.
Style	The required style for the group. This can be either of the following values:
Title	The new group title.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

UIMS.BORDER	Enclose the group in a box.
UIMS.NONE	Do not enclose the group in a box.

Comments

If the group has no bounding box, the title will not be displayed.

InitialiseUims

This subroutine initialises the UIMS environment.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL InitialiseUims

Comments

This subroutine must be called at the start of an application, before any other UIMS subroutines are used.

InitialiseUims calls the **IsUimsCapable** subroutine and sets the COMMON variable **UIMS.CAPABLE** to the result.

See Also

IsUimsCapable.

IsUimsCapable

This subroutine returns whether or not the terminal supports UIMS.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL IsUimsCapable(*vCapable*)

Syntax Elements

vCapable A variable in which to return whether or not the terminal supports UIMS. This will be one of the following values:

TRUE	The terminal supports UIMS.
FALSE	The terminal does not support UIMS.

Comments

This subroutine allows the programmer to determine whether or not the terminal supports UIMS, without calling **InitialiseUims**.

If UIMS has already been initialised, interrogating the COMMON **UIMS.CAPABLE** variable is quicker than calling **IsUimsCapable**.

ListBoxAddContent – ListBoxAddSelections

These subroutines change the attributes of a **ListBox** contact.

- **ListBoxAddContent** adds a single item to the contents of a list box.
- **ListBoxAddContents** adds a group of items to the contents of a list box.
- **ListBoxAddSelection** marks an item within the list box as selected.
- **ListBoxAddSelections** marks multiple items within the list box as selected.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ListBoxAddContent(*Context*, *ListBox*, *Index*, *Item*, *vErr*)

CALL ListBoxAddContents(*Context*, *ListBox*, *Index*, *aItemList*, *vErr*)

CALL ListBoxAddSelection(*Context*, *ListBox*, *Selection*, *vErr*)

CALL ListBoxAddSelections(*Context*, *ListBox*, *aSelectList*, *vErr*)

Syntax Elements

Context	The handle of the application context.
ListBox	The handle of the **ListBox** contact.
Index	The point in the list of contents at which the new items are to be added. The list is numbered starting from 0 and new items are added before the specified existing item. An index of -1 adds the new item to the end of the list.
Item	A string containing the text of the item which is to be added to the list box contents.
aItemList	A dynamic array containing the items that are to be added to the contents of the list box, each item consisting of a text string that will be displayed in the list box.
Selection	The position of the item to be selected within the list box. The list is numbered starting from zero.

aSelectList	A dynamic array containing a list of indexes into the list box contents. The items in this list will become marked as selected. The list is numbered starting from zero.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

If the list box allows only one selection, **ListBoxAddSelection** and **ListBoxAddSelections** will cancel any previous selections.

If multiple selections are attempted in a list box that allows only one selection at a time, only the first selection in the list will be made.

See Also

ListBoxGetContent, **ListBoxGetContents**, **ListBoxRemoveContent**, **ListBoxRemoveContents**, **ListBoxGetSelections**, **ListBoxRemoveSelection**, **ListBoxRemoveSelections**.

ListBoxGetContent – ListBoxGetSelections

These subroutines return the different attributes of a **ListBox** contact.

- **ListBoxGetContent** returns the text of one item from a list box.
- **ListBoxGetContents** returns a list of all the items in a list box.
- **ListBoxGetSelections** returns the indexes of the currently selected items.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ListBoxGetContent(*Context*, *ListBox*, *Index*, *vItem*, *vErr*)

CALL ListBoxGetContents(*Context*, *ListBox*, *vaItemList*, *vErr*)

CALL ListBoxGetSelections(*Context*, *ListBox*, *vaSelectList*, *vErr*)

Syntax Elements

Context	The handle of the application context.
ListBox	The handle of the **ListBox** contact.
Index	The position in the list of the first item whose text you require. The list is numbered starting from 0.
vItem	A variable in which to return the requested item.
vaItemList	A variable in which to return the requested items. If there is more than one item, this variable will be returned as a dynamic array, with one item in each attribute.
vaSelectList	A variable in which to return the indexes of the selected items. If there is more than one item selected, this variable will be returned as a dynamic array, with one index number in each attribute.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

ListBoxAddContent, **ListBoxAddContents**, **ListBoxRemoveContent**, **ListBoxRemoveContents**, **ListBoxSetLink**, **ListBoxAddSelection**, **ListBoxAddSelections**, **ListBoxRemoveSelection**, **ListBoxRemoveSelections**.

ListBoxRemoveContent – ListBoxRemoveSelections

These subroutines change the attributes of a **ListBox** contact.

- **ListBoxRemoveContent** deletes a named item from the contents of the list box.
- **ListBoxRemoveContents** deletes a number of items from the list box, starting at a specified position.
- **ListBoxRemoveSelection** marks an item within the list box as not selected.
- **ListBoxRemoveSelections** marks items within the list box as not selected.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ListBoxRemoveContent(*Context*, *ListBox*, *Item*, *vErr*)

CALL ListBoxRemoveContents(*Context*, *ListBox*, *Index*, *Count*, *vErr*)

CALL ListBoxRemoveSelection(*Context*, *Listbox*, *Selection*, *vErr*)

CALL ListBoxRemoveSelections(*Context*, *Listbox*, *aSelectList*, *vErr*)

Syntax Elements

Context	The handle of the application context.
ListBox	The handle of the **ListBox** contact.
Item	The text of the item to be deleted from the list.
Index	The position in the list of contents at which to start deleting items. The list is numbered starting from 0.
Count	The number of items to be deleted from the list. To remove every item from the starting point (*Index* parameter) to the end of the list, specify a count of -1.
Selection	The position in the list of contents of the item which is to be deselected.
aSelectList	A dynamic array containing the positions in the list of contents of the items to be deselected.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

ListBoxAddContent, **ListBoxAddContents**, **ListBoxGetContent**, **ListBoxGetContents**, **ListBoxAddSelection**, **ListBoxAddSelections**, **ListBoxGetSelections**.

ListBoxSetLink

This subroutine links a list box to an **EditBox** contact. A selection made in the list box will then be automatically copied into the edit box.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ListBoxSetLink(*Context*, *ListBox*, *EditBox*, *vErr*)

Syntax Elements

Context	The handle of the application context.
ListBox	The handle of the **ListBox** contact.
EditBox	The handle of the **EditBox** contact to which the list box is to be linked.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

LoadAppRes

This subroutine creates the objects and contacts defined in a compiled UIMS resource file.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL LoadAppRes(*Context*, *FileName*, *vErr*)

Syntax Elements

Context	The application context.
Filename	A string containing the name of the resource file. If no path is specified, the file is loaded from the disk and directory specified in the RFW.INI file on the PC.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

LoByte

This subroutine returns the value of the least-significant byte of a word (2 byte) value.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL LoByte(*Word*, *vLoByte*)

Syntax Elements

Word	The word value from which you require the least-significant byte.
vLoByte	A variable in which to return the value of the least-significant byte.

Comments

LoByte allows the programmer to determine the values of the individual bytes in the composite word values returned by the **GetMsg** subroutine.

Word will normally be a composite value returned by **GetMsg**.

Example

The following fragment of code determines whether any keyboard modifier keys (SHIFT, CTRL, ALT) were held down when a mouse button was clicked.

```
* Wait for the next message
CALL GetMsg(0, ...
               CONTEXT, ...
               WINDOW, ...
               CONTACT, ...
               MSGTYPE, ...
               TIMESTAMP, ...
               DATA1, ...
               DATA2, ...
               DATA3, ...
               DATA4, ...
               ERR)

BEGIN CASE
CASE MSGTYPE = UIMS.MSG.CLICK
```

```
* Use LoByte to separate out the modifier keys
    CALL LoByte(DATA1, MOD)
    IF MOD THEN
        PRINT "A keyboard modifier was held down."
    END ELSE
        PRINT "No keyboard modifiers were held down."
    END

END CASE
```

See Also

GetMsg, **HiByte**.

MakePullDownMenu

This subroutine creates a complete menu, including all its menu items.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL MakePullDownMenu(*Context*, *Ident*, *Title*, *FirstItem*, *aItemTitles*, *Parent*, *vMenu*)

Syntax Elements

Context	The handle of the application context.
Ident	An integer value to use as the handle for the **Menu** contact. If this parameter is zero, a handle will be assigned by UIMS. In either case, the handle of the newly-created menu is returned in the *vMenu* parameter.
Title	The title of the menu. An ampersand (&) preceding a character in this string denotes that character as the selector key for the menu.
FirstItem	An integer value to use as the handle for the first **MenuItem** contact on the menu. UIMS will assign handles to the remaining menu items by incrementing this value by one for each subsequent menu item. If the *FirstItem* parameter is zero, UIMS assign a handle for the first menu item, by incrementing the handle of the **Menu** contact by one.
aItemTitles	A dynamic array with each attribute containing the title of one of the items on the menu. If a character in the string is preceded by an ampersand (&), that character is assigned as the selector key for the menu item. If a single hyphen is used as the title, a separator item is created. This appears as a continuous line across the width of the menu. A separator item cannot be selected by the user and should be used to visually group related menu items. Note that a separator item cannot be attached to a menu bar.
Parent	The handle of the parent of the menu, if required. If specified, this must be either a **MenuBar** or another **Menu**. If the parent is currently displayed the menu will be drawn immediately. If *Parent* is a null string, the contact is created without a parent and can be attached at a later time using **AddChild** or **AddChildren**.
vMenu	A variable in which to return the handle of the newly-created **Menu**. If it could not be created for any reason, zero is returned. Note, however, that if

asynchronous error handling is selected and a handle has been supplied in the *Ident* parameter, this handle will always be returned, and any error will be reported by means of a **UIMS.MSG.NOTIFY** message. See **SetSync** for more details.

Cascading Menus

MakePullDownMenu can also be used to create cascading menus – menus that are children of other menus, rather than of the menu bar. This is done by creating the parent menu with space reserved for the child menu, and then filling this reserved space when creating the child menu.

Space for a child menu is reserved by including a null title in the appropriate position in the *aItemTitles* array. No **MenuItem** contact will be created, but the corresponding handle will be set aside for later use. When **MakePullDownMenu** is used to create the child menu, the *Ident* parameter must specify the handle assigned to the reserved space. Note, however, that unless the reserved item is the last item on the parent menu, when creating the child the *FirstItem* parameter cannot be zero; this is because, when **MakePullDownMenu** adds one to *Ident* to create the handle for the child's first menu item, this handle will be the same as that already assigned to the next item on the parent menu.

Note: The parent and child menus can be created in any order. If the child menu already exists when its parent is created, the child will simply be inserted into its reserved space.

Example

The following example creates a Format menu with Character, Paragraph and Border items, and two cascaded menus: Tabs and Page.

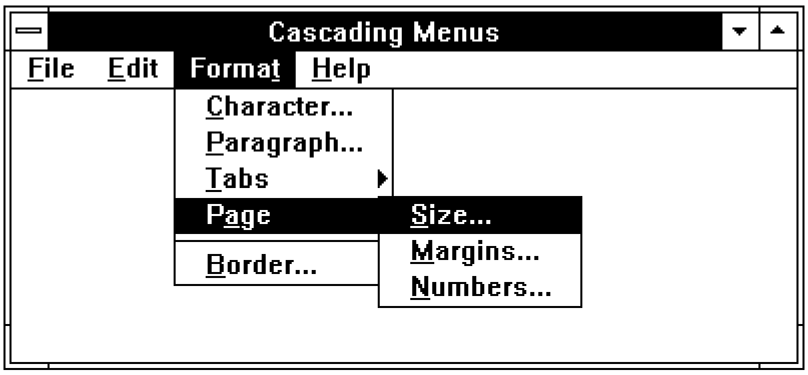
```
FORMATITEMS = "&Character..." ;* id = 101
FORMATITEMS<-1> = "&Paragraph..." ;* id = 102
FORMATITEMS<-1> = "" ;* tabs id = 103
FORMATITEMS<-1> = "" ;* page id = 104
FORMATITEMS<-1> = "-" ;* separator id = 105
FORMATITEMS<-1> = "&Border..." ;* id = 106

TABITEMS = "&Set..." ;* id = 200
TABITEMS<-1> = "&Clear..." ;* id = 201
TABITEMS<-1> = "&Reset all" ;* id = 202

PAGEITEMS = "&Size..." ;* id = 300
PAGEITEMS<-1> = "&Margins..." ;* id = 301
PAGEITEMS<-1> = "&Numbers..." ;* id = 302

CALL MakePullDownMenu(CONTEXT, ...
                      100, ...
```

```
                            "&Format", ...
                            0, ...
                            FORMATITEMS, ...
                            0, ...
                            FILE)
CALL MakePullDownMenu(CONTEXT, ...
                            103, ...
                            "&Tabs", ...
                            200, ...
                            TABITEMS, ...
                            File, ...
                            FILETABS)
CALL MakePullDownMenu(CONTEXT, ...
                            104, ...
                            "&Page", ...
                            300, ...
                            PAGEITEMS, ...
                            File, ...
                            FILEPAGE)
```



Comments UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

See Also **CreatePullDownMenu**, **CreateMenuBar**, **CreateMenuItem**.

Map

This subroutine makes a contact mappable; that is, it makes it possible to display the contact on the screen.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Map(*Context*, *Contact*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A mappable contact will only be visible if it has a parent and that parent is visible. Making a contact with a visible parent mappable will make it and any mappable children visible.

Newly-created contacts are mappable.

See Also

UnMap, **SetMapped**, **GetState**.

MenuItemCheck

This subroutine displays a check mark beside a **MenuItem**.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL MenuItemCheck(*Context*, *MenuItem*, *vErr*)

Syntax Elements

Context	The handle of the application context.
MenuItem	The handle of the **MenuItem** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The type of check mark displayed is platform dependent. On a PC running Microsoft Windows, a tick (✓) is used.

See Also

MenuItemSetCheckMark, **MenuItemUncheck**, **MenuItemGetCheckMark**, **MenuItemSetAutoCheck**.

MenuItemGetCheckMark

This subroutine returns whether or not a check mark is displayed beside a **MenuItem** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL MenuItemGetCheckMark(*Context*, *MenuItem*, *vCheck*)

Syntax Elements

Context	The handle of the application context.
MenuItem	The handle of the **MenuItem** contact.
vCheck	A variable in which to return whether or not the menu item is checked. This will be one of the following values:

TRUE	The menu item is checked.
FALSE	The menu item is not checked.

See Also

MenuItemSetAutoCheck, **MenuItemSetCheckMark**, **MenuItemSetTitle**.

MenuItemSetAutoCheck – MenuItemSetTitle

These subroutines set different attributes of a **MenuItem** contact.

- **MenuItemSetAutoCheck** sets whether checking and unchecking the menu item is automatic or not.
- **MenuItemSetCheckMark** checks or unchecks a menu item.
- **MenuItemSetTitle** changes the title of a menu item.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL MenuItemSetAutoCheck(*Context*, *MenuItem*, *Autocheck*, *vErr*)

CALL MenuItemSetCheckMark(*Context*, *MenuItem*, *Check*, *vErr*)

CALL MenuItemSetTitle(*Context*, *MenuItem*, *Title*, *vErr*)

Syntax Elements

Context The handle of the application context.

MenuItem The handle of the **MenuItem** contact.

Autocheck Specifies whether to automatically check and uncheck or not. This must be one of the following values:

TRUE	Enable automatic checking.
FALSE	Disable automatic checking.

Check Specifies whether the menu item is to become checked or unchecked. This must be one of the following values:

TRUE	Check the menu item.
FALSE	Uncheck the menu item.

Note that the type of check mark displayed is platform dependent. On a PC running Microsoft Windows, a tick (✓) is used.

Title	The new title to be displayed for the menu item. An ampersand (&) preceding a character in this string denotes that character as the selector key for the menu. If a single hyphen is used as the title, a separator item is created. This appears as a continuous line across the width of its parent menu. A separator item cannot be selected by the user and should be used to visually group related menu items. Note that a separator item cannot be attached to a menu bar.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

MenuItemCheck, **MenuItemUncheck**, **CreateMenuItem**, **MenuItemGetCheckMark**.

MenuItemUncheck

This subroutine removes the check mark (if any) displayed beside a **MenuItem** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL MenuItemUncheck(*Context*, *MenuItem*, *vErr*)

Syntax Elements

Context	The handle of the application context.
MenuItem	The handle of the **MenuItem** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

MenuItemCheck, **MenuItemSetCheckMark**, **MenuItemGetCheckMark**.

MenuSetTitle

This subroutine changes the title of a **Menu** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL MenuSetTitle(*Context*, *Menu*, *Title*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Menu	The handle of the **Menu** contact.
Title	The new title to be displayed for the menu. An ampersand (&) preceding a character in this string denotes that character as the selector key for the menu.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreatePullDownMenu, **MakePullDownMenu**.

Move

This subroutine changes the position of a contact within its parent.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Move(*Context*, *Contact*, *HPos*, *VPos*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact you wish to move.
HPos	The new horizontal position for the contact in coordinate units.
VPos	The new vertical position for the contact in coordinate units.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

HPos and *VPos* specify the position of the top left-hand corner of the contact, relative to the top left-hand corner of its parent's client area (position 0,0). Note, however, the following exceptions:

- For contacts that are children of the application context, the position must be specified relative to the top, left-hand corner of the display (position 0,0).
- For **ExclusiveGroup** and **InclusiveGroup** contacts, the top of the contact is aligned with the top of the title text (even if none is displayed), rather than with the top of the enclosing box.

The position specified is interpreted in accordance with the coordinate mode (text or graphics) currently selected for the application context.

Provided the contact is mappable, when it is moved it will always be redrawn immediately.

See Also

GetPosition.

OptionButtonDeselect

This subroutine deselects the specified **OptionButton** contact, clearing the check mark if one is displayed.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL OptionButtonDeselect(*Context*, *Button*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **OptionButton** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

OptionButtonSelect, **OptionButtonSetSelected**, **OptionButtonGetSelected**.

OptionButtonGetSelected

This subroutine returns the current state (selected or deselected) of an **OptionButton** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL OptionButtonGetSelected(*Context*, *Button*, *vSelected*)

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **OptionButton** contact.
vSelected	A variable in which to return whether or not the button is selected. This will be one of the following values:

TRUE	The button is selected.
FALSE	The button is not selected.

See Also

OptionButtonSetSelected.

OptionButtonSelect

This subroutine selects the specified **OptionButton**.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL OptionButtonSelect(*Context*, *Button*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **OptionButton** contact.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

When a option button is selected its check circle is displayed filled in.

See Also

OptionButtonDeselect, **OptionButtonSetSelected**, **OptionButtonGetSelected**.

OptionButtonSetSelected – OptionButtonSetToggle

These subroutines change the attributes of a specified **OptionButton** contact.

- **OptionButtonSetSelected** sets the button to selected or deselected.
- **OptionButtonSetTitle** changes the title displayed beside the button.
- **OptionButtonSetToggle** changes the auto-toggle state of the button.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL OptionButtonSetSelected(*Context*, *Button*, *Selected*, *vErr*)

CALL OptionButtonSetTitle(*Context*, *Button*, *Title*, *vErr*)

CALL OptionButtonSetToggle(*Context*, *Button*, *Toggle*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **OptionButton** contact.
Selected	The required button state. This must be one of the following values:

TRUE	Select the button.
FALSE	Deselect the button.

Title	The new title for the button.
Toggle	The required auto-toggle state. This must be one of the following values:

TRUE	Enable automatic toggling.
FALSE	Disable automatic toggling.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

When a option button is selected its check circle is displayed filled in.

See Also

OptionButtonGetSelected.

Paste

This subroutine pastes the contents of the clipboard into an **EditBox** or **TextEditor** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Paste(*Context*, *Contact*, *Character*, *Line*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
Character	The character position at which to paste the data. The position must be specified as the number of characters from the start of the line specified in *Line*.
Line	The number of the line containing the position at which to paste the data. If *Contact* is the handle of an **EditBox**, this parameter will be ignored.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

If *Character* and *Line* are both -1, the Clipboard contents are inserted at the current cursor position, replacing any selected text.

If *Contact* is the handle of a contact other than an **EditBox** or **TextEditor**, an error will be returned.

See Also

ClipboardGetContent, **ClipboardGetState**, **Copy**, **Cut**, **ClipboardSetContent**.

PenGetColour, PenGetWidth

These subroutines return the different attributes of a **Pen** object.

- **PenGetColour** returns the colour of the pen.
- **PenGetWidth** returns the width of the pen.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL PenGetColour(*Context*, *Pen*, *vColour*)

CALL PenGetWidth(*Context*, *Pen*, *vWidth*)

Syntax Elements

Context	The handle of the application context.
Pen	The handle of the **Pen** object.
vColour	A variable in which to return the colour of the pen. The value returned will be a UIMS logical colour or an RGB value (see Appendix B).
vWidth	A variable in which to return the width, in pixels, of the pen.

See Also

PenSetColour, **PenSetWidth**.

PenSetColour, PenSetWidth

These subroutines change the attributes of a specified **Pen** object.

- **PenSetColour** changes the colour of the pen.
- **PenSetWidth** changes the width of the pen.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL PenSetColour(*Context*, *Pen*, *Colour*, *vErr*)

CALL PenSetWidth(*Context*, *Pen*, *Width*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Pen	The handle of the **Pen** object.
Colour	The colour of the pen. This must be a UIMS logical colour or an RGB value (see Appendix B).
Width	The width, in pixels, of lines drawn by the pen. If the width is set to zero, the pen will draw the thinnest and/or most efficient lines available on the display platform.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

PenGetColour, **PenGetWidth**.

PointerGetType

This subroutine returns the shape of a **Pointer** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL PointerGetType(*Context*, *Pointer*, *vType*)

Syntax Elements

Context	The handle of the application context.
Pointer	The handle of the **Pointer** object.
vType	A variable in which to return the shape of the pointer. This will be one of the following values:

UIMS.PTR.ARROW	Standard arrow pointer.
UIMS.PTR.IBEAM	Text I-beam pointer.
UIMS.PTR.CROSS	Diagonal cross-hair pointer.
UIMS.PTR.PLUS	Horizontal and vertical cross-hair pointer.
UIMS.PTR.WAIT	Wait pointer - normally an hourglass.

See Also

PointerSetType.

PointerSetType

This subroutine changes the shape of a specified **Pointer** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL PointerSetType(*Context*, *Pointer*, *Type*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Pointer	The handle of the **Pointer** object.
Type	The shape of the pointer. This must be one of the following values:

UIMS.PTR.ARROW	Standard arrow pointer.
UIMS.PTR.IBEAM	Text I-beam pointer.
UIMS.PTR.CROSS	Diagonal cross-hair pointer.
UIMS.PTR.PLUS	Horizontal and vertical cross-hair pointer.
UIMS.PTR.WAIT	Wait pointer - normally an hourglass.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreatePointer, **PointerGetType**.

ReMapNVLine25

This subroutine allows you to use a UIMS message box to display system messages which the host sends to line 25 of the terminal screen.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL ReMapNVLine25(*Context*, *Enable*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Enabled	The required state. This must be one of the following values:

TRUE	Display messages in a message box.
FALSE	Display messages on line 25.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The message box in which host system messages are displayed has a single OK button, and therefore requires a response from the user.

Applications which use line 25 for a continuous display of status information should not map system messages to a message box.

See Also

CreateMessageBox.

RemoveChild, RemoveChildren

These subroutines remove objects from another object's list of children.

- **RemoveChild** removes a particular child from the list.
- **RemoveChildren** removes a number of children from the list, starting at a specified position.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL RemoveChild(*Context*, *Object*, *Child*, *vErr*)

CALL RemoveChildren(*Context*, *Object*, *Index*, *Count*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Object	The handle of the parent object.
Child	The handle of the child you wish to remove.
Index	The position in the list of the child or children to be removed. The list is numbered starting from 0.
Count	The number of children to be removed. To remove every child from the starting point (*Index* parameter) to the end of the list, specify a count of -1.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

AddChild, **AddChildren**, **GetChild**, **GetChildren**, **GetObjectParent**.

RemoveTimer

This subroutine removes a timer which was created with **AddTimer**.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL RemoveTimer(*Context*, *Timer*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Timer	The handle of the timer.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

AddTimer.

Resize

This subroutine changes the size of a contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Resize(*Context*, *Contact*, *Width*, *Height*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact whose size you wish to change.
Width	The required contact width in coordinate units.
Height	The required contact height in coordinate units.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

For **ExclusiveGroup** and **InclusiveGroup** contacts, this subroutine sets the overall size of the contact including any title text, rather than the size of the enclosing box.

The *Width* and *Height* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

Provided the contact is mappable, when it is resized it will always be redrawn immediately.

An App or Child window has a minimum size, which depends on the style and content of the window. Any attempt to make either the width or height smaller than the minimum will fail.

See Also

GetSize.

Scroll

This subroutine scrolls the client area of the specified window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL Scroll(*Context, Window, HScroll, VScroll, Left, Top, Right, Bottom, vErr*)

Syntax Elements

Context	The handle of the application context.
Window	The window which is to be scrolled.
HScroll	The amount of horizontal movement. If *HScroll* is positive, the contents of the client area move to the right, relative to the window border; if *HScroll* is negative, the contents of the client area move to the left.
VScroll	The amount of vertical movement. If *VScroll* is positive, the contents of the client area move downwards, relative to the window border; if *VScroll* is negative, the contents of the client area move upwards.
Left	The position of the left-hand edge of the area to be scrolled.
Top	The position of the top edge of the area to be scrolled.
Right	The position of the right-hand edge of the area to be scrolled.
Bottom	The position of the bottom edge of the area to be scrolled.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Left, Top, Right and *Bottom* permit only a part of the client area to be scrolled; if all are zero, the entire client area will be scrolled. If specified, the edges of the scrolled area are relative to the top left-hand corner of the window's client area (position 0,0).

The *HScroll, VScroll, Left, Top, Right* and *Bottom* parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

If the window does not have a text canvas, text data which is scrolled out of the window is lost and, if re-displayed, must be redrawn by the application.

ScrollBarGetThumb

This subroutine returns the value corresponding to the current thumb position of a **ScrollBar** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ScrollBarGetThumb(*Context*, *ScrollBar*, *vPosition*)

Syntax Elements

Context	The handle of the application context.
ScrollBar	Handle of the **ScrollBar** object
vPosition	A variable in which to return the value corresponding to the current position of the thumb.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

ScrollBarSetInc, **ScrollBarSetRange**, **ScrollBarSetThumb**, **ScrollBarSetTracking**.

ScrollBarSetInc – ScrollBarSetTracking

These subroutines change the different attributes of a **ScrollBar** contact.

- **ScrollBarSetInc** sets the increments by which the thumb position value is changed when line and page scrolling are used.
- **ScrollBarSetRange** sets the minimum and maximum thumb position values, corresponding to the opposite ends of the thumb track.
- **ScrollBarSetThumb** moves the scroll-bar thumb.
- **ScrollBarSetTracking** changes the scroll-bar thumb tracking mode.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL ScrollBarSetInc(*Context*, *ScrollBar*, *PageInc*, *LineInc*, *vErr*)

CALL ScrollBarSetRange(*Context*, *ScrollBar*, *Min*, *Max*, *vErr*)

CALL ScrollBarSetThumb(*Context*, *ScrollBar*, *Position*, *vErr*)

CALL ScrollBarSetTracking(*Context*, *ScrollBar*, *Track*, *vErr*)

Syntax Elements

Context	The handle of the application context.
ScrollBar	The handle of the scroll-bar.
PageInc	The required value for the page scroll increment.
LineInc	The required value for the line scroll increment.
Min	A value corresponding to the top or left-hand end of the thumb track.
Max	A value corresponding to the bottom or right-hand end of the thumb track.
Position	A value representing the required thumb position.

Track	The required tracking mode for the scroll-bar. This must be one of the following values:

TRUE	Enable tracking.
FALSE	Disable tracking.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

If the *Min* parameter is greater than *Max* an error is returned.

See Also

CreateScrollBar, **ScrollBarGetThumb**.

SendKeys

Sends a sequence of keypresses to the active Windows application, as if they had been typed at the keyboard.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE RFWKEYS FROM UIMS-TOOLS

CALL SendKeys(*Keys*, *Control*, *vErr*)

Syntax Elements

Keys	A string containing a key sequence.
Control	One of the following control settings:

SENDKEYS.WAIT	Don't send the keypresses until the next call to the **Execute** subroutine.
RFW.NONE	Send the keypresses immediately.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will be set to **ERR.RFW.SUCCESS** for successful completion or will contain one of the SendKeys error codes listed in Appendix D.

Non-Printable Characters

When **SendKeys** is used in an application, the item RFWKEYS must be included from the file UIMS-TOOLS. This item contains key definitions for non-printable characters as listed in Table 6-1.

Table 6-1. SendKeys Key Definitions

Key	Code	Key	Code
ALT GR	**SVK.ALTGRKEY**	CAPS LOCK	**SVK.CAPSLOCK**
ALT	**SVK.ALTKEY**	CTRL	**SVK.CTRLKEY**
Backspace	**SVK.BKSP**	DELETE	**SVK.DEL**
BREAK (CTRL+PAUSE)	**SVK.BREAK**	DOWN (cursor key)	**SVK.DOWN**
END	**SVK.END**	Function key F9	**SVK.F9**
RETURN	**SVK.ENTER**	Keypad minus (-)	**SVK.GREYMINUS**

(continued)

Table 6-1 SendKeys Key Definitions (continued)

Key	Code	Key	Code
ESC	**SVK.ESC**	Keypad plus (+)	**SVK.GREYPLUS**
Function key F1	**SVK.F1**	HOME	**SVK.HOME**
Function key F10	**SVK.F10**	INSERT	**SVK.INS**
Function key F11	**SVK.F11**	LEFT (cursor key)	**SVK.LEFT**
Function key F12	**SVK.F12**	SHIFT (left)	**SVK.LSHIFTKEY**
Function key F13	**SVK.F13**	NUM LOCK	**SVK.NUMLOCK**
Function key F14	**SVK.F14**	PAUSE	**SVK.PAUSE**
Function key F15	**SVK.F15**	PAGE DOWN	**SVK.PGDN**
Function key F16	**SVK.F16**	PAGE UP	**SVK.PGUP**
Function key F2	**SVK.F2**	PRINT SCREEN	**SVK.PRTSC**
Function key F3	**SVK.F3**	RIGHT (cursor key)	**SVK.RIGHT**
Function key F4	**SVK.F4**	SHIFT (right)	**SVK.RSHIFTKEY**
Function key F5	**SVK.F5**	SCROLL LOCK	**SVK.SCROLL**
Function key F6	**SVK.F6**	TAB	**SVK.TAB**
Function key F7	**SVK.F7**	UP (cursor key)	**SVK.UP**
Function key F8	**SVK.F8**		

In addition, the following key modifiers are available:

ALT	**SVK.ALT**
ALT GR	**SVK.ALTGR**
CTRL	**SVK.CTRL**
SHIFT	**SVK.SHIFT**

Key Rate

The following allow you to control the rate at which the keys are sent:

- **SVK.KEYRATE:***time***:***terminator*

  where

:	is the DATA/BASIC string concatenation operator.

time	is the time between keys, in 18ths of a second.
terminator	is any non-numeric character.

- **SVK.DELAY:***delay***:***terminator*

  where

:	is the DATA/BASIC string concatenation operator.
delay	is the time to pause before sending the next key, in seconds.
terminator	is any non-numeric character.

Examples

```
CALL SendKeys("s", RFW.NONE, ERR)
```

Sends a lower case 's' without waiting for the next **Execute** call.

```
* check that the file exists
FILE = "C:\TMP\SKTEXT.TXT"
CALL SystemCommand(SYS.EXIST, RFW.NONE, FILE, RESPONSE, ERR)

IF ERR = ERR.SYS.SUCCESS THEN
       KEYS = SVK.KEYRATE:9:"." ;* Set the key rate
       CALL SendKeys(KEYS, SENDKEYS.WAIT, ERR)

       * ALT+E,A - select the whole file
       KEYS = SVK.ALT:"ea"
       * CTRL+INSERT - copy to clipboard
       KEYS = KEYS:SVK.CTRL:SVK.INS
       * ALT+F4 - close Notepad
       KEYS = KEYS:SVK.ALT:SVK.F4
       CALL SendKeys(KEYS, SENDKEYS.WAIT, ERR)

       COMMANDLINE = "NOTEPAD.EXE ":FILE
       CALL Execute(COMMANDLINE, ...
                    EXECUTE.SHOWMAXIMIZED, ...
                    EXECUTE.WAIT, ...
                    ERR)
END
```

Tests for the existence of the file C:\TMP\SKTEXT.TXT and then, if it exists, builds up a key sequence which sets a key rate of one every half second, selects the entire contents of the file, places it on the Windows clipboard, and then closes the application. Finally the

Windows Notepad utility is executed with the file SKTEXT.TXT loaded, and the stored key sequence is sent to this application.

Comments

Only one instance of key replay can occur at a time. A applications that use this facility must be programmed to handle the **ERR.SENDKEYS.INUSE** error when calling the **SendKeys** and **Execute** functions.

If required, successive calls to **SendKeys** can be used to build up a sequence of keys, before sending them all with a single **Execute** call.

Note: Use great care when sending keys to other programs. UIMS has no way to detect or correct errors generated by other programs, and always sends the programmed series of keystrokes. Make sure that you test your program under a variety of conditions to ensure that the keystrokes required by the other program remain exactly the same. If the other program requires different keystrokes to those programmed, data could be lost. If more keystrokes are required than are programmed (for instance, if a RETURN is required to respond to a message box), the program will freeze while waiting for the missing input. If there are too many keystrokes in the programmed sequence the results will be unpredictable.

See Also

Execute.

SetBorderStyle

This subroutine changes the border style of an App or Child window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetBorderStyle(*Context*, *Contact*, *Style*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the window.
Style	The new border style. This must be one of the following values:

UIMS.BORDER	Give the window a border.
UIMS.NONE	No border.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

GetBorderStyle, **CreateAppWin**, **CreateChildWin**.

SetClip

This subroutine sets the boundaries of a window's clipping region.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetClip(*Context*, *Window*, *Top*, *Left*, *Bottom*, *Right*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Window	The handle of the window.
Top	The position, in coordinate units, of the top edge of the clipping region.
Left	The position, in coordinate units, of the left-hand edge of the clipping region.
Bottom	The position, in coordinate units, of the bottom edge of the clipping region.
Right	The position, in coordinate units, of the right-hand edge of the clipping region.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Top, *Left*, *Bottom* and *Right* must be specified relative to the top left-hand corner of the window's client area (position 0,0). These parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

Setting *Top*, *Left*, *Bottom* and *Right* all to zero removes any previously set clipping region. With no clipping region set, text and graphics will be clipped at the edges of the client area, whatever its size.

See Also

GetClip.

SetContactFocus

This subroutine gives the focus to a particular contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetContactFocus(*Context*, *Contact*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact which is to receive the focus.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Some contacts cannot accept the focus. If the specified contact has children, the focus will normally pass to the first child in its list of children which can accept the focus. If the contact has no children, an error will be returned.

See Also

GetChildFocus.

SetCoordMode

This subroutine sets the coordinate mode by which positions on the screen are referenced.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetCoordMode(*Context*, *CoordMode*, *vErr*)

Syntax Elements

Context	The handle of the **AppContext** object.
CoordMode	The required coordinate mode. This must be one of the following values:

UIMS.COORD.TEXT	Screen positions are referenced to the nearest character position, where the size of a character is that of an upper case character in the default system typeface.
UIMS.COORD.GRAPHIC	Screen positions are referenced to the nearest pixel.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

When an application signs on to UIMS, text mode is selected.

See Also

GetCoordMode.

SetCursorPosition, SetCursorState

These subroutines change the different attributes of the cursor within an **AppWindow** or **ChildWindow** contact.

- **SetCursorPosition** changes the position of the text cursor within the window.
- **SetCursorState** sets the type of text cursor that is currently selected and whether or not the cursor is visible.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetCursorPosition(*Context*, *Window*, *HPos*, *VPos*, *vErr*)

CALL SetCursorState(*Context*, *Window*, *Visible*, *CurType*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Window	The handle of the **AppWindow** or **ChildWindow** contact.
HPos	The horizontal coordinate of the cursor position.
VPos	The vertical coordinate of the cursor position.
Visible	Specifies whether or not the cursor is visible. This must be one of the following values:

TRUE	Make the cursor visible.
FALSE	Hide the cursor.

CurType	A value representing the type of cursor displayed. This will be one of the following:

UIMS.BAR	Vertical bar.
UIMS.BLOCK	Block cursor.
UIMS.OUTLINE	Outline cursor.
UIMS.UNDERLINE	Underline cursor.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

HPos and *VPos* must be specified relative to the top left-hand corner of the window's client area (position 0,0). These parameters will be interpreted according to the coordinate mode (text or graphics) currently selected for the application context.

See Also

GetCursorPosition, **GetCursorState**.

SetDrawrule

This subroutine attaches a new **Drawrule** object to the specified object or contact. This changes attributes such as foreground and background colour (refer to the description of the **Drawrule** object in Chapter 3).

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetDrawrule(*Context*, *Object*, *Drawrule*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Object	The handle of the object or contact to which the drawrule is to be attached.
Drawrule	The handle of the **Drawrule** object. If this parameter is zero, the current drawrule will be removed. Note, however, that if the contact has a parent, the old drawrule will be replaced by that attached to the parent object.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A drawrule can be attached to only the following objects and contacts:

AppWindow
ChildWindow
Line
Rectangle
Text
AppContext

Attempting to attach a drawrule to an object or contact other than those listed above will result in an error.

See Also

GetDrawrule.

SetEnabled

This subroutine enables or disables a contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetEnabled(*Context*, *Contact*, *Enabled*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact to be enabled or disabled.
Enabled	The required state. This must be one of the following values:

TRUE	Enable the contact.
FALSE	Disable the contact.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A disabled contact remains displayed, but cannot be selected by the user. The disabled state is indicated a greying effect, the exact form of which is platform dependent.

See Also

Enable, **Disable**, **GetState**.

SetEnabledNVGroup

This subroutine enables or disables a NewView group.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMSDEFS FROM UIMS-TOOLS ;* Only required for contact groups
INCLUDE UIMSCOMMON FROM UIMS-TOOLS ;* Only required for contact groups

CALL SetEnabledNVGroup(*Context*, *Group*, *Enable*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Group	The identifier for the group to be enabled or disabled.
Enabled	The required state. This must be one of the following values:

TRUE	Enable the group.
FALSE	Disable the group.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A disabled group remains displayed, but cannot be selected by the user. In the case of groups of contacts, the disabled state is indicated a greying effect, the exact form of which is platform dependent. For groups of hot-spots, the mouse pointer does not change shape as it passes over them.

See Also

SetMappedNVGroup.

SetEventMask

This subroutine specifies which types of message will be received by the application. A mask can be applied to the whole application, or to individual objects.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetEventMask(*Context*, *Object*, *EventMask*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Object	The handle of an object.
EventMask	The new event mask for the object. This must be a combination of the event mask constants listed in Chapter 4.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

An event mask specifies which types of message will be passed on from an object to its parent. Since all objects and contacts are ultimately children of the application context, an event mask applied to the **AppContext** object controls which types of message will be received by the application.

See Also

GetEventMask, **SetSecondaryEventMask**.

SetHelpFile – SetHelpKey

These subroutines change the settings of the application's **AppHelp** object.

- **SetHelpFile** attaches a help file on the PC to the application.
- **SetHelpIndex** associates a contact with a section of the help file.
- **SetHelpKey** assigns a key as the help accelerator.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetHelpFile(*Context*, *Filename*, *vErr*)

CALL SetHelpIndex(*Context*, *Contact*, *Section*, *vErr*)

CALL SetHelpKey(*Context*, *Key*, *vErr*)

Syntax Elements

Context	The handle of the **AppContext**.
Filename	A string containing the name of the help file. If no path is specified, the file is loaded from the disk and directory specified in the RFW.INI file on the PC.
Contact	The handle of a contact.
Section	The help-id of the section of the help file that is to be associated with the specified contact.
Key	The virtual key code of the key that is to be assigned as the help accelerator.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

GetHelpFile, **GetHelpIndex**, **GetHelpKey**, **SetNVHelp**.

SetMapped

This subroutine allows you to decide whether or not a contact is displayed on the screen.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetMapped(*Context*, *Contact*, *Mapped*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact.
Mapped	The required state. This must be one of the following values:
	TRUE Make the contact visible (mappable).
	FALSE Make the contact invisible (unmappable).
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A mappable contact will only be visible if it has a parent and that parent is visible.

- Making a contact with a visible parent mappable will make it and any mappable children visible.

- Conversely, making a contact with a visible parent unmappable will make it and any children invisible.

Newly-created contacts are mappable.

Menu and **MenuItem** contacts cannot be made unmappable.

See Also

Map, **UnMap**, **GetState**.

SetMappedNVGroup

This subroutine allows you to decide whether or not a NewView group is displayed on the screen.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS
INCLUDE UIMSDEFS FROM UIMS-TOOLS ;* Only required for contact groups
INCLUDE UIMSCOMMON FROM UIMS-TOOLS ;* Only required for contact groups

CALL SetMappedNVGroup(*Context*, *Group*, *Mapped*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Group	The identifier for the group.
Mapped	The required state. This must be one of the following values:

TRUE	Make the group visible.
FALSE	Make the group invisible.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The effect of this subroutine depends on whether the group consists of contacts or hot-spots, and also on the type of contact:

- For button contacts, setting them mappable makes them visible; setting them unmappable makes them invisible. Since an invisible contact cannot be operated by the user, unmappable buttons are in effect also disabled.

- **MenuItem** contacts cannot be made unmappable, so this subroutine will have no effect on this type of contact.

- For hot-spots, setting them mappable makes them visible by drawing a border around them; setting them unmappable makes them invisible. However, unlike button contacts, invisible hot-spots can still be operated by the user – to disable the hot-spots use the **SetEnabledNVGroup** subroutine.

See Also

SetEnabledNVGroup.

SetNVHelp

This subroutine attaches a help file on the PC to a NewView application.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL SetNVHelp(*Filename, vErr*)

Syntax Elements

Filename	A string containing the name of the help file. If no path is specified, the file is loaded from the disk and directory specified in the RFW.INI file on the PC.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

This subroutine allows a NewView application to provide application-specific help. The file specified is displayed when the user selects the Application (or equivalent) command from the RealLink Help menu. Refer to Chapter 5 for more details.

See Also

SetHelpFile.

SetPointer

This subroutine attaches a new **Pointer** object to the specified object or contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetPointer(*Context*, *Object*, *Pointer*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Object	The handle of the object or contact to which the pointer is to be attached.
Pointer	The handle of the **Pointer** object.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

A pointer can be attached to only the following objects and contacts:

AppWindow
ChildWindow
AppContext

Attempting to attach a pointer to an object or contact other than those listed above will result in an error.

See Also

GetPointer.

SetPointerPos

This subroutine sets the position of the mouse pointer, relative to either the screen or a specified contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetPointerPos(*Context*, *Contact*, *HPos*, *VPos*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of a contact. If this parameter is zero the position is set to the screen.
HPos	The new horizontal position for the pointer in coordinate units.
VPos	The new vertical position for the pointer in coordinate units.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

SetPointerPos sets the position of the pointer's hot-spot. If a contact is specified, the position is set relative to the top left-hand corner of the contact's client area (position 0,0); otherwise the position is set relative to the top left-hand corner of the screen.

The position specified is interpreted in accordance with the coordinate mode (text or graphics) currently selected for the application context.

See Also

GetPointerPos.

SetSecondaryEventMask

This subroutine sets a secondary event mask for an application.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetSecondaryEventMask(*Context*, *EventMask*, *Unmaskable*, *Alert*, *vErr*)

Syntax Elements

Context The handle of the application context.

EventMask The secondary event mask for the application. This must be a combination of the event mask constants listed in Chapter 4.

Unmaskable This specifies whether messages which cannot be masked should be allowed to reach the application. This must be one of the following values:

TRUE	Allow non-maskable messages to reach the application.
FALSE	Prevent non-maskable messages reaching the application.

Alert This parameter is reserved for future use – it must be set to **FALSE**.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The secondary event mask is described in Chapter 4.

See Also

GetSecondaryEventMask, **SetEventMask**.

SetSync

This subroutine selects synchronous or asynchronous error response handling for UIMS subroutine calls.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetSync(*Context*, *Mode*, *vErr*)

Syntax Elements

Context The handle of the application context.

Note: In UIMS version 2.0 this parameter is reserved for future use – any value will be ignored.

Mode The required error handling mode. This must be one of the following values:

TRUE	Synchronous.
FALSE	Asynchronous.

vErr This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Unless changed, UIMS handles errors asynchronously.

Synchronous and asynchronous error handling are described on page 6-3.

See Also

GetMsg.

SetTeFontSize

This subroutine sets the point size for text displayed in the RealLink or currently active 'terminal emulation' (TE) window.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetTeFontSize(*PointSize***,** *Flags***,** *vErr***)**

Syntax Elements

PointSize	The required point size. This should one of those which is available for use in the RealLink or TE window – use **GetTePointSizes** to find out which sizes are available. If a size that is not available is requested, an error is returned.
Flags	This parameter is reserved for future use – it must be set to zero.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The point size currently in use in the RealLink or TE window can be obtained by calling **GetTeFontSize**.

See Also

GetTeFontSize, **GetTeFontSizes**.

SetTeWindow

This subroutine changes the window that is used as the application's 'terminal emulation' (TE) window – that is the window in which output printed to the terminal (using PRINT, CRT, etc.) will be displayed.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetTeWindow(*Context*, *Window*, *Controls*, *Err*)

Syntax Elements

Context The handle of the application context. If this parameter is zero, the current RealLink context is assumed.

Window The handle of the App or Child window that is to act as the TE window. If this parameter is zero, the terminal emulation function is returned to the RealLink window.

Controls Determines whether or not the RealLink window is to remain visible, and the characteristics of the new TE window. This must be a combination of the following values:

TE.SHOWWIN	The RealLink window is to remain visible. If not set, the RealLink window will be hidden.
TE.NOAUTOSCROLL	Disables the horizontal scroll bar. If not set, a horizontal scroll bar will appear should the TE window become too narrow to display 80 characters at the current point size.
TE.NOAUTORESIZE	Disables the RealLink Auto Resize Window feature (if selected for the RealLink window).
TE.NOAUTOFONT	Disables the RealLink Auto Select Font feature (if selected for the RealLink window).
TE.10PTFONT	Changes the text point size to 10pt. If not set, the point size currently selected for the RealLink window will be retained.

The following pre-defined style is also available:

UIMS.NONE	None of the above.

In general, when setting a new TE window the RealLink window should be hidden (do not select the **TE.SHOWWIN** option). When the terminal emulation function is returned to the RealLink window (*Window* = 0), **TE.SHOWWIN** must be selected.

vErr — This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

The screen coordinate mode (see **SetCoordMode**) must be set before calling **SetTeWindow**. Changing the coordinate mode once the TE window is set is not recommended, but should this be done, **SetTeWindow** must be called again, even if the TE window is to remain unchanged.

When a UIMS application is run, the RealLink window will remain visible unless hidden by **SetTeWindow**. Applications which do not change the TE window can hide the RealLink window by calling **SetTeWindow** with the *Context* and *Window* parameters both set to zero, and **TE.SHOWWIN** not selected. For example:

```
CALL SetTeWindow(0, 0, UIMS.NONE, ERR)
```

Before leaving the application and returning to RealLink, the RealLink window must be re-displayed as follows:

```
CALL SetTeWindow(0, 0, TE.SHOWWIN, ERR)
```

See Also

GetTeFontSize, **GetTeFontSizes**, **SetTeFontSize**.

SetUimsMode

This subroutine restores message processing after calls to:

- NewView subroutines.
- the **Execute**, **SendKeys**, or **SystemCommand** subroutines.
- DATA/BASIC commands that send data to or receive data from the terminal.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetUimsMode

Comments

This subroutine must be used before calling **GetMsg**, if any subroutines or commands of the types listed above have been used. If this is not done, Keyboard messages will be ignored.

SetUpdate

This subroutine allows you to specify when a contact will be redrawn if a change occurs.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SetUpdate(*Context*, *Contact*, *Update*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact whose update mode you wish to change.
Update	The update mode you require for the contact; this must be one of the following values:

UIMS.IMMEDIATE	Redraw immediately.
UIMS.NONE	Don't redraw; wait for a **Draw** command.

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Some operations (for example, **Move**, **Resize**) occur immediately whatever the update mode.

The update mode of **Menu** and **MenuItem** contacts is always the same as the **MenuBar** to which they are attached and cannot be changed independently. If *Contact* is the handle of a **Menu** or a **MenuItem**, **SetUpdate** returns an error.

See Also

GetUpdate, **Draw**.

SignOff

This subroutine signs off a UIMS session.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SignOff(*Context*, *vErr*)

Syntax Elements

Context	The handle of the **AppContext** object that is to be signed off.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

When this subroutine is called, UIMS destroys any remaining objects created during the session. The session is then terminated.

See Also

SignOn.

SignOn

This subroutine signs on a UIMS session and creates an **AppContext** object for the new session.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SignOn(*AppName, vContext*)

Syntax Elements

AppName	A string containing the name of the application.
vContext	A variable in which to return the handle of the newly-created **AppContext** object. If the sign on was not successful, the handle returned will be zero.

Comments

The subroutine must be called before any of the other UIMS subroutines can be used.

See Also

SignOff.

SoundSpeaker

This subroutine sounds the loudspeaker in the PC.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL SoundSpeaker(*Pitch*, *Duration*, *Repeat*, *Delay*, *vErr*)

Syntax Elements

Pitch	The frequency in Hertz of the required sound.
Duration	The duration of the sound in milliseconds.
Repeat	The number of repeats required.
Delay	The time delay between repeats.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

StartImage

Loads the image manager, thus permitting the use of the **DisplayImage** and **EraseImage** subroutines.

Syntax

INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL StartImage(*vImageMan*)

Syntax Elements

vImageMan	A variable in which to return the handle of the image manager. If the Image Manager could not be loaded for any reason, zero is returned.

Comments

The **StopImage** routine *must* be called to unload the image manager before closing the application.

See Also

DisplayImage, **EraseImage**, **StopImage**.

StopImage

Unloads the image manager. Once this has been done, the **DisplayImage** and **EraseImage** subroutines can no longer be used.

Syntax

INCLUDE UIMSCOMMON FROM UIMS-TOOLS
INCLUDE UIMS-DDE FROM UIMS-TOOLS

CALL StopImage(*ImageMan*, *vErr*)

Syntax Elements

ImageMan	The handle of the image manager, returned by the **StartImage** subroutine.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. A return value of zero indicates successful completion. Otherwise, one of the error codes listed in Appendix D is returned.

Note: **StopImage** errors are always returned synchronously. **UIMS.MSG.NOTIFY** messages are not generated (see page 6-3).

Comments

This routine *must* be called before closing the application.

See Also

DisplayImage, **EraseImage**, **StartImage**.

SystemCommand

Runs a DOS system command on the PC.

Syntax

INCLUDE RFWDEFS FROM UIMS-TOOLS

CALL SystemCommand(*CommandCode*, *Options*, *ParamString*, *vResponse*, *vErr*)

Syntax Elements

CommandCode	A value representing the required system command.
Options	A value which specifies command options.
ParamString	A string containing command parameters. This can contain any combination of characters, except for "%" which must be used to enclose substitutable parameters (see below).
vResponse	A variable in which to return the result of the command.
vErr	A variable in which to return the completion status of the subroutine. In most cases this will be **ERR.SYS.SUCCESS** for success, **ERR.SYS.FAIL** for failure, or **ERR.SYS.INVCOMMAND** for an invalid command (but see below).

Commands

The following commands are available:

SYS.CREATEDIR

Creates a directory on the PC.

Options	None. Must be set to zero.
ParamString	The name for the new directory. This must not contain ambiguous characters (* or ?).
vResponse	Returned set to a null string.
vErr	Returned set to **ERR.SYS.SUCCESS** for success or **ERR.SYS.FAIL** for failure.

SYS.DELDIR

Deletes a directory on the PC.

Options	None. Must be set to zero.

ParamString	The name of the directory to delete. This must not contain ambiguous characters (* or ?).
vResponse	Returned set to a null string.
vErr	Returned set to **ERR.SYS.SUCCESS** for success or **ERR.SYS.FAIL** for failure.

Notes:

1. A directory can only be deleted if it is empty.

2. It is not possible to delete the root directory or the current working directory.

SYS.DELFILE Deletes a file on the PC.

Options	None. Must be set to zero.
ParamString	The name of the file to delete. This must not contain ambiguous characters (* or ?).
vResponse	Returned set to a null string.
vErr	Returned set to **ERR.SYS.SUCCESS** for success or **ERR.SYS.FAIL** for failure.

SYS.DOSEXEC Starts a DOS or Windows program on the PC.

Options None. Must be set to zero.

ParamString A string containing the name of the program, plus any optional parameters and/or switches. The program name can be that of a PIF file. If the program name does not contain a directory path, UIMS will search the PC for the executable file as follows:

1. The currently selected directory on the PC.

2. The directories listed in the PATH environment variable.

vResponse Returned set to a null string.

vErr	Returned set to **ERR.SYS.SUCCESS** for success or to one of the Execute error codes listed in Appendix D.

SYS.EXIST

Checks whether a file or directory exists on the PC and, if required, returns information about that file or directory.

Options The information required. This must be a combination of the following values:

EXIST.TIMESTAMP	Return the date and time that the file or directory was last modified.
EXIST.SIZE	Return the size of the file in bytes.
EXIST.HEADER	Return the first line of the file.
RFW.NONE	Check whether the file or directory exists, but do not return any information about it.

ParamString The name of a file or directory. This must not contain ambiguous characters (* or ?).

vResponse A dynamic array containing the result of the command. Each attribute contains the result of one of the selected options, in the order **EXIST.TIMESTAMP**, **EXIST.SIZE**, **EXIST.HEADER**. Only the results of selected options are returned.

The results of the different options are as follows:

EXIST.TIMESTAMP
A string in the format:

weekday month day hour : *min* : *sec year* X'0D' X'0A'.

For example: `Wed Jan 02 04:26:55 1992`

EXIST.SIZE
The size of the file in bytes.

EXIST.HEADER
A string containing up to 36 characters from the beginning of the file. Only printable characters [CHAR(32) to CHAR(127)] are returned –

the string ends at the first non-printable character or after 36 characters, whichever is the sooner.

If no options are selected, a null string is returned.

vErr Returned set to one of the following values:

ERR.SYS.SUCCESS
A file with the specified name exists.
ERR.SYS.DIRECTORY
A directory with the specified name exists.
ERR.SYS.NOFILE
The file or directory does not exist.

SYS.LOOKUP Performs any substitutions in the *ParamString* parameter and returns the result.

Options	None. Must be set to zero.
ParamString	A command parameter string containing substitutable parameters (see below).
vResponse	Returned containing the string which results from replacing any substitutable parameters in *ParamString*.
vErr	Always returned set to **ERR.SYS.SUCCESS**.

Substitutable Parameters

The *ParamString* parameter can contain substitutable parameters enclosed in percent signs. The following substitutions are available:

%*EnvVar***%**
where *EnvVar* is the name of a DOS environment variable.

The percent signs and the text in between are replaced by the contents of the specified variable. For example, `%path%` is replaced by the DOS executable search path.

%*Section***!***Key***!***Default***!***IniFile***%**
where

Section	is the name of a section in the INI file specified in *IniFile* (see below). The default value is "reallink".
Key	is a the name specific parameter within that section.

Default	is the string to be returned if the specified key cannot be found. The default is a null string.
IniFile	is the name of a Windows INI file. The default value is "RFW.INI".

For example, `%!rfwdir!!%` is replaced by the name of the RealLink for Windows program directory, and `%intl!iCountry!!WIN.INI%` is replaced by the current Windows country code.

%%
is replaced by a single percent sign.

Notes:

1. After substitution, any pairs of backslashes ("\\") are converted to single backslashes ("\").

2. Substitutable parameters that do not conform to the above rules are removed from *ParamString*.

Examples

```
CALL SystemCommand(SYS.CREATEDIR, 0, "c:\uimstemp", RESPONSE, ERR)
```

Creates a directory called `c:\uimstemp` on the PC.

```
CALL SystemCommand(SYS.DELDIR, 0, "c:\uimstemp", RESPONSE, ERR)
```

Deletes the directory called `c:\uimstemp` from the PC.

```
CALL SystemCommand(SYS.DELFILE, ...
                   0, ...
                   "c:\uimstemp\myfile.txt", ...
                   RESPONSE, ...
                   ERR)
```

Deletes the file called `c:\uimstemp\myfile.txt` from the PC.

```
CALL SystemCommand(SYS.EXIST, ...
                   EXIST.SIZE + EXIST.HEADER, ...
                   "%!resourcepath!!%\generic.res", ...
                   RESPONSE, ...
                   ERR)
```

Checks whether the file `generic.res` exists on the PC in RealLink's resource directory and, if it does, returns a dynamic array containing its size (in the first attribute) and its first 36 bytes (in the second attribute).

```
CALL SystemCommand(SYS.LOOKUP, 0, "%!helppath!!%", RESPONSE, ERR)
```

Returns the directory on the PC that contains the RealLink help files.

See Also

Execute.

TextEditorGetContent, TextEditorGetTextLen

These subroutines return the different attributes of a **TextEditor** contact.

- **TextEditorGetContent** returns the text from the text editor.
- **TextEditorGetTextLen** returns the length of the text.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL TextEditorGetContent(*Context*, *Editor*, *vText*, *vErr*)

CALL TextEditorGetTextLen(*Context*, *Editor*, *vLength*)

Syntax Elements

Context	The handle of the application context.
Editor	The handle of the **TextEditor** contact.
vText	A variable in which to return the contents of the text editor. The text will be returned as a null terminated string with attribute marks separating the individual lines.
vLength	A variable in which to return the number of text characters. Note that the attribute marks separating multiple lines are included in the count.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

TextEditorSetContent.

TextEditorSetContent

This subroutine assigns a text string to a **TextEditor** contact for editing or display.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL TextEditorSetContent(*Context*, *Editor*, *Text*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Editor	Handle of the **TextEditor** object
Text	The text string to be displayed for editing in the text editor window. The text can consist of one or more lines, with multiple lines separated by attribute marks.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

CreateTextEditor, **TextEditorGetContent**.

TextGetContent

This subroutine returns the text displayed in a **Text** contact.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL TextGetContent(*Context*, *Text*, *vString*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Text	The handle of the **Text** contact.
vString	A variable in which to return the text string.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

TextSetContent.

TextSetContent, TextSetJustification

These subroutines change the different attributes of a **Text** contact.

- **TextSetContent** changes the text displayed.
- **TextSetJustification** changes the alignment of the text.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL TextSetContent(*Context*, *Text*, *String*, *vErr*)

CALL TextSetJustification(*Context*, *Text*, *Just*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Text	The handle of the **Text** contact.
String	The new text string.
Just	The alignment of the text. This must be one of the following values:
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

UIMS.JUST.LEFT	Left aligned.
UIMS.JUST.RIGHT	Right aligned.
UIMS.JUST.BOTH	Both left and right aligned (justified).
UIMS.JUST.CENTRED	Centred.

See Also

CreateText, **TextGetContent**.

TitledButtonSetStyle, TitledButtonSetTitle

These subroutines change the different attributes of a **TitledButton** contact.

- **TitledButtonSetStyle** changes the style of the button.
- **TitledButtonSetTitle** changes the title displayed inside the button.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL TitledButtonSetStyle(*Context*, *Button*, *Style*, *vErr*)

CALL TitledButtonSetTitle(*Context*, *Button*, *Title*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Button	The handle of the **TitledButton** contact.
Style	The required style for the button. This must be one of the following values:
Title	The new button title.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Values for *Style*:

UIMS.NONE	Normal (thin) border.
UIMS.TB.THICK	Thickened border **-** indicates a default button.

See Also

CreateTitledButton.

TypeFaceGetName – TypeFaceGetPointSizes

These subroutines return different attributes of a **TypeFace** object.

- **TypeFaceGetName** returns the name of the typeface.
- **TypeFaceGetPointSize** returns one of the available point sizes.
- **TypeFaceGetPointSizes** returns a list of the available point sizes for the typeface.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL TypeFaceGetName(*Context*, *TypeFace*, *vFontName*, *vErr*)

CALL TypeFaceGetPointSize(*Context*, *TypeFace*, *Index*, *vPointSize*)

CALL TypeFaceGetPointSizes(*Context*, *TypeFace*, *vaPointsizes*, *vErr*)

Syntax Elements

Context	The handle of the application context.
TypeFace	The handle of a **TypeFace** object.
vFontName	A variable in which to return the name of the typeface.
Index	The position in the list of the point size you require. The list is numbered starting from 0.
vPointSize	A variable in which to return the point size. If zero is returned, there is no point size at the requested position.
vaPointsizes	A variable in which to return a list of numbers representing the available point sizes. The list is returned as a dynamic array with one point size in each attribute.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

GetTypeFace, **GetTypeFaces**.

UngrabPointer

This subroutine releases the pointer following a call to **GrabPointer**. See page 6-152 for details.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL UngrabPointer(*vErr*)

Syntax Elements

vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

See Also

GrabPointer.

UnMap

This subroutine makes a contact unmappable; that is, it removes the contact from the screen.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL UnMap(*Context*, *Contact*, *vErr*)

Syntax Elements

Context	The handle of the application context.
Contact	The handle of the contact you wish to remove from the screen.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

Making a contact unmappable also removes its children from the screen. The mapped state of the children, however, remains unchanged.

See Also

Map, **SetMapped**, **GetState**.

WaitPointerOff

This subroutine changes the mouse pointer from the wait pointer to the pointer type specified by the **Pointer** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL WaitPointerOff(*Context*, *vErr*)

Syntax Elements

Context	The handle of the **AppContext**.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

This subroutine should be called following a call to **WaitPointerOn**, to restore the pointer type to its previous state.

Note that **WaitPointerOff** also performs an **UngrabPointer**.

See Also

WaitPointerOn, **PointerSetType**, **UngrabPointer**.

WaitPointerOn

This subroutine changes the mouse pointer to a wait pointer (normally an hourglass), overriding the pointer type specified by the **Pointer** object.

Syntax

INCLUDE UIMSDEFS FROM UIMS-TOOLS
INCLUDE UIMSCOMMON FROM UIMS-TOOLS

CALL WaitPointerOn(*Context*, *vErr*)

Syntax Elements

Context	The handle of the **AppContext**.
vErr	This is a variable that must be supplied to return the completion status of the subroutine. It will contain a UIMS error code if an error has occurred, or will be zero for successful completion.

Comments

This subroutine should be called to change the appearance of the mouse pointer while lengthy processing is in progress. When this processing is complete, the **WaitPointerOff** subroutine should be called to restore the pointer type to that specified by the **Pointer** object.

Note that **WaitPointerOn** also performs a **GrabPointer**.

See Also

WaitPointerOff, **PointerSetType**, **GrabPointer**.

Chapter 7
Resource Compiler

This chapter describes how to use the UIMS Resource Compiler to create resource files on the PC.

Introduction

The UIMS resource compiler allows the graphical objects used by an application to be defined on the PC rather than the host. This has two advantages:

- Processing is shared between the PC and the host, reducing communication between the two systems and therefore improving performance.
- Resources created in this way are loaded only when the application is run, allowing a programmer to produce different versions of an application, without having to change the host program.

Resources are defined in a source file (resource script) on the PC; this can be produced by any ASCII text editor (Windows Notepad, for example). The completed source must then be compiled, using the UIMS Resource Compiler, RLRC. A host UIMS application loads the compiled resources by calling the **LoadAppRes** subroutine.

Object Definitions

Each object must be defined by a statement with the form

```
OBJECTTYPE = Ident
{
        ATTRIBUTE = Value
        [ATTRIBUTE = Value
        …]
}
```

where:

OBJECTTYPE	is the type of object being defined: SCROLLBAR, EDITBOX, etc. - see the list of object types in Appendix C. This must be in upper case.
Ident	is an integer value by which the object will be identified. Once the object has been loaded into the UIMS application, this value will be used as a handle.

If the value 0 is used, UIMS will assign a handle to the object. Note, however, that there is no way of discovering the value of this handle, and that this should therefore only be done for objects to which the application will never require access. Typical examples are separators in a menu and static text in a dialog box.

Note: UIMS reserves handles 8000 to 9999 for its own use – these must not be used by the application.

ATTRIBUTE	is an attribute of the object: SIZE, ENABLED, etc. This must be in upper case.
Value	is the value to be assigned to this attribute. This can be any of the following:

- A numeric value; for example: 10.
- A valid Resource Compiler keyword for the specified attribute; for example: TRUE.

- A literal string, enclosed in single quotes; for example: 'Title'. If the string itself contains a single quote, this must be preceded by another single quote; for example: 'Type ' 'c' ' to continue'.

- A list of similar or dissimilar settings (depending on the attribute) separated by commas; for example: CHILDREN 200, 221, 234, 378

Example

The following example defines a titled button called Cancel. The button is 60 coordinate units wide by 18 high and is positioned 80 coordinate units across and 24 down relative to its parent. The title displayed in the button is Cancel.

```
TITLEDBUTTON = 102
{
       SIZE = 60, 18
       POSITION = 80, 24
       TITLE = 'Cancel'
}
```

Any number of attributes may be specified between the braces. Note however, that the braces must always be present.

Nested Definitions

Object definitions can be nested in order to associate children with their parents. For example, a dialog box might be designed with three option buttons and two titled buttons. In order to automatically associate the buttons with the window, a definition of the following form would be needed:

```
DIALOGBOX = 200
{
       TITLE = 'Choose a sentence'
       POSITION = 200,300
       SIZE = 266,62
       STYLE = CLOSABLE,MOVABLE

       OPTIONBUTTON = 201
       {
              TITLE = 'Hello'
              POSITION = 10,6
              SIZE = 68,16
       }
```

```
        OPTIONBUTTON = 202
        {
                TITLE = 'Good-bye'
                POSITION = 10,23
                SIZE = 96,16
        }

        OPTIONBUTTON = 203
        {
                TITLE = 'Good morning'
                POSITION = 10,40
                SIZE = 133,16
        }

        TITLEDBUTTON = 221
        {
                TITLE = 'OK'
                POSITION = 167,7
                SIZE = 88,21
        }

        TITLEDBUTTON = 222
        {
                TITLE = 'CANCEL'
                POSITION = 167,31
                SIZE = 88,21
        }
}
```

The same result can be achieved by defining the option and titled buttons separately and then specifying the CHILDREN attribute for the dialog window. Note, however, that if nesting is not used, the children must be defined before their associated parent object, in order that the parent can be created successfully. If this is not done, the resource compilation will fail. The following example shows this alternative method:

```
OPTIONBUTTON = 201
{
        TITLE = 'Hello'
        POSITION = 10,6
        SIZE = 68,16
}
```

```
OPTIONBUTTON = 202
{
      TITLE = 'Good-bye'
      POSITION = 10,23
      SIZE = 96,16
}

OPTIONBUTTON = 203
{
      TITLE = 'Good morning'
      POSITION = 10,40
      SIZE = 133,16
}

TITLEDBUTTON = 221
{
      TITLE = 'OK'
      POSITION = 167,7
      SIZE = 88,21
}

TITLEDBUTTON = 222
{
      TITLE = 'CANCEL'
      POSITION = 167,31
      SIZE = 88,21
}

DIALOGBOX = 200
{
      TITLE = 'Choose a sentence'
      POSITION = 200,300
      SIZE = 266,62
      STYLE = CLOSABLE,MOVABLE
      CHILDREN = 201,202,203,221,222
}
```

Menus

Additional features are available when defining **MenuBar** and **Menu** contacts.

MenuItem Definition

Within the definition of a menu bar or menu, simple menu items can be automatically defined as part of the CHILDREN attribute. There is no need for separate menu item definitions, either nested within or separate from the definition of the parent.

To create menu items in this way, the CHILDREN attribute must be defined as a list of menu item titles, each followed by an equals sign and an identifying number. Each menu item title must be enclosed in single quotes. For example, in a MENU called Edit, there might be items such as Cut, Paste, Delete. These would be coded as follows:

```
MENU = 250
{
      TITLE = 'Edit'
      CHILDREN = 'Cut'=251,'Paste'=252,'Delete'=253
}
```

The Resource Compiler will create a **MenuItem** contact for each entry in the list of children. The example given above would be equivalent to:

```
MENUITEM = 251
{
      TITLE = 'Cut'
}

MENUITEM = 252
{
      TITLE = 'Paste'
}

MENUITEM = 253
{
      TITLE = 'Delete'
}

MENU = 250
{
      CHILDREN = 251,252,253
}
```

MenuItem Attributes

When a menu item is defined in a CHILDREN statement as described above, certain attributes can be set by including additional characters in the title text.

& (ampersand) Designates the following character as a selector key. When the menu item is displayed, the character concerned is shown underlined and the user can select the item by pressing that key.

Notes:

1. You can also use an ampersand in this way when defining Menu contacts.

2. The user must press the ALT key to activate the menu bar before using a selector key to select an item from the menu bar.

! (exclamation mark)
This causes the menu item to be checked – equivalent to setting the CHECKMARK attribute to TRUE. For example:

```
MENU = 75
{
      TITLE = 'View'
      CHILDREN = 'Normal!'=80,'Draft'=81,'Page'=82
}
```

defines a View menu with Normal, Draft and Page items. The Normal item is checked.

+ (plus sign) Causes the menu item to be disabled (greyed) – equivalent to setting the ENABLED attribute to FALSE. For example:

```
MENU = Edit
{
      TITLE = 'Edit'
      CHILDREN = 'Cut+','Paste','Delete+'
}
```

defines an Edit menu with Cut, Paste and Delete items. The Cut and Delete items are disabled.

Note that these substitutions only apply to the CHILDREN attribute of **MenuBar** and **Menu** definitions.

Separator Items

If a single hyphen is used as the title of a menu item, a separator item is created. This appears as a continuous line across the width of its parent menu. A separator item cannot be selected by the user and should be used to visually group related menu items. Note that a separator item cannot be attached to a menu bar.

Screen Coordinates

Most types of contact have Size and Position attributes which can be set in the resource script. When the resources are loaded into an application, these attributes are interpreted in accordance with the coordinate mode (text or graphics) set for the application:

- If text mode is selected, the coordinates are interpreted as character positions, based on the average size of the upper case characters in the default system font.

- If graphics mode is selected, the coordinates are interpreted as pixel positions on an arbitrary screen 1000 pixels wide by 1000 pixels high. When the resources are loaded into an application, the coordinates are scaled to fit within the actual screen.

  For example, if an application is displayed on a VGA screen (640 pixels wide by 480 pixels high), and the resource file specifies position 500, 500 for a contact, when the resource is loaded into the application, the horizontal coordinate will be converted to position 320 and the vertical coordinate to position 240 – that is half-way across and half-way down the screen. Similarly, if the size of a contact is specified as 250 pixels wide by 750 high, when loaded and displayed, it will always be one quarter of the screen wide and three quarters of the screen high, whatever the screen resolution.

Resource File Control

The first line of the source file may contain the version number in the form.

VERSION = *string*

This will be written after the first record of the output file. If it is not the first line of the source file, it will be ignored.

Comments

Comments may be included in the source code at any point except within literal strings or in the middle of a word. A comment can be indicated in three ways:

- It can be placed between the characters '/*' and '*/', as in the C programming language. For example:

```
/* These two lines
   form a single comment */
```

- It can be placed on a separate line which starts with an asterisk (*), as in DATA/BASIC. For example:

```
* This is a comment
```

- It can be placed on the end of a line, if preceded by the characters ';*', as in DATA/BASIC. For example:

```
TITLEDBUTTON = 102 ;* Cancel button
```

White-space Characters

Spaces, newline characters and tab characters may be used freely within the source code to aid readability. They will be ignored by the compiler. Note, however, that a single line cannot be longer than 200 characters.

Pre-processor Commands

The UIMS Resource Compiler includes a pre-processor which manipulates the text of a source file as the first phase of compilation. Pre-processor commands are typically used to make resource files easy to change and easy to compile for different execution environments. Commands in the source file tell the pre-processor to perform specific actions. For example, the pre-processor can replace identifiers in the text, insert the contents of other files into the source file, or suppress some definitions by removing sections of the text.

The pre-processor recognises the following commands:

#DEFINE
EQUATE
EQU
#INCLUDE
#IFDEF
#ELSE
#ENDIF

A pre-processor command will only be recognised if it occurs at the beginning of a line - if the command is preceded by spaces or tabs, it will be ignored. Note that, with the exception of the EQUATE and EQU commands which must always be in upper case, these commands can be in upper case, as shown, or in lower case, as used in C language program and header files.

Constant Definitions

Constants may be used to associate meaningful identifiers (tokens) with values and keywords. A token can be redefined as many times as required within the source code, the new value applying only to code which follows the re-definition.

A constant can be defined in three ways:

#DEFINE *token*[*value*]

EQUATE *token* **TO** *value*

EQU *token* **TO** *value*

where

#DEFINE, EQUATE and **EQU**
are the three forms of the pre-processor command. Note that in the case of #DEFINE, this can be in upper case as shown, or in lower case as used in C language program and header files. The EQUATE and EQU keywords must be in upper case.

token	is an identifier which is used later in the source code.
TO	is an additional keyword required by the EQUATE and EQU forms of the command. This must be in upper case.
value	is the value which will replace the identifier wherever it is found in the source code following this definition. It may be a number, a text string (enclosed in single quotes) or another identifier.

If no substitution is required (for instance, when defining tokens to be used by the #IFDEF statement – see below), the #DEFINE form can be used with no *value* parameter.

Example

```
#DEFINE HEAD 'Heading to be used for all windows'

#DEFINE Win1 100
#DEFINE Win2 150

APPWINDOW = Win1
{
      TITLE = HEAD
         .
         .
         .
}

APPWINDOW = Win2
{
      TITLE = HEAD
         .
         .
         .
}
```

In this example, the pre-processor will replace every occurrence of the token HEAD with the text 'Heading to be used for all windows', and the contact names Win1 and Win2 with the identifiers 100 and 150 respectively.

File Inclusion

The #INCLUDE command inserts the contents of a named file into the source code. You can create files which contain constant definitions and then use #INCLUDE commands to add these definitions to any source file.

#INCLUDE tells the pre-processor to treat the contents of the named file as if it appeared in the source at the point where the command appears. The included text can also contain pre-processor commands and these are carried out before processing of the original source file resumes. An included file can itself contain #INCLUDE commands, up to a maximum of 5 levels.

The syntax of the #INCLUDE command is as follows:

#INCLUDE [*drive***:**][*path*] *filename*

where

#INCLUDE	is the pre-processor command.
[*drive***:**][*path*] *filename*	specifies the file to be included.

Conditional Compilation

If required, the same source file can be used to generate different versions of an application. Directives are provided which allow you to suppress compilation of parts of a source file by testing a constant expression or identifier to determine which text blocks should be removed from the source file during pre-processing.

The syntax of these directives is as follows:

#IFDEF *ident*
    source code block
[**#ELSE**
    source code block]
#ENDIF

where *ident* is an identifier which might have been previously defined by a constant-definition pre-processor command. If *ident* has been defined, regardless of its value, the source code lines immediately following the #IFDEF statement are included in the source to be compiled and those following the #ELSE statement (if any) are removed. If *ident* has not been defined, the source code following the #ELSE statement (if any) is included instead.

Source code blocks can include both object definitions and pre-processor directives.

#IFDEF statements can be nested within each other, up to a maximum of 9 levels. An #ELSE statement is always assumed to be associated with the most recent open #IFDEF statement. Consider the following,

```
#IFDEF Ident1
       Block1
#IFDEF Ident2
       Block2
#ELSE
       Block3
#ENDIF
#ENDIF
```

The blocks which are compiled depend on the states of *Ident1* and *Ident2* as follows:

Ident1	***Ident2***	**Blocks compiled**
		None
Defined		*Block1*, *Block3*
	Defined	None
Defined	Defined	*Block1*, *Block2*

However, in

```
#IFDEF Ident1
       Block1
#IFDEF Ident2
       Block2
#ENDIF
#ELSE
       Block3
#ENDIF
```

the following applies:

Ident1	***Ident2***	**Blocks compiled**
		Block3
Defined		*Block1*
	Defined	*Block3*
Defined	Defined	*Block1*, *Block2*

Compiling a Resource Script

A resource script source file is compiled by using the RLRC command. This has the following syntax:

RLRC [*filename*]

If you omit the *filename* parameter, you will be prompted for the name of your source file:

```
Resource script filename (.ucl) :
```

The source-file name supplied as input to the RLRC command must have the suffix '.UCL' (UIMS Command Language). Files included with the #INCLUDE pre-processor command can have the suffixes '.UCL' or '.H'.

The compilation process creates an output file with the same name as the source file, but with the suffix '.RES'.

Note: The resource compiler can be run from any directory, but must have access to the files RC.DAT and RC.MSG. These files must be in the directory specified by the DOS environment variable URCPATH. If this variable is not set, the files are assumed to be in the current directory.

If you intend to run RLRC from directories other that containing RC.DAT and RC.MSG, you should set URCPATH to the correct directory. For example.

```
SET URCPATH=C:\RFW
```

tells the resource compiler that RC.DAT and RC.MSG are in the directory C:\RFW.

If required, URCPATH can be set at boot time by including the above command in the AUTOEXEC.BAT file.

Errors

When a compilation error occurs, the action taken depends on whether it has been detected by the pre-processor or the compiler.

- If the error occurs during pre-processing, an error message is displayed and the line containing the error is ignored.

- If the error occurs during compilation, the number of the line in which the error occurred is displayed, together with an error message. All subsequent source lines are ignored, up to the closing brace of the current outer nested level. Compilation then continues from this point.

Note that the line numbers reported are not those in the original source file, but in a temporary file, RCTEMP, created in the current directory. If errors occur, this file should be examined to determine their location.

Example

The following example illustrates how errors are reported.

The file RESOURCE.UCL, shown below, contains two errors:

❶ This line contains an incomplete pre-processor command.

❷ A mandatory **OptionButton** attribute – SIZE – has been commented out.

```
EQU Dialog TO 200
EQU Hi TO 201
EQU Bye TO 202
EQU Morning TO 203
EQU OK ❶
EQU Cancel TO 222

OPTIONBUTTON = Hi
{
       TITLE = 'Hello'
*      SIZE = 68,16 ❷
       POSITION = 10,6
}

OPTIONBUTTON = Bye
{
       TITLE = 'Good-bye'
       POSITION = 10,23
       SIZE = 96,16
}

OPTIONBUTTON = Morning
{
       TITLE = 'Good morning'
       POSITION = 10,40
       SIZE = 133,16
}
```

```
TITLEDBUTTON = OK
{
       TITLE = 'OK'
       POSITION = 167,7
       SIZE = 88,21
}

TITLEDBUTTON = Cancel
{
       TITLE = 'CANCEL'
       POSITION = 167,31
       SIZE = 88,21
}

DIALOGBOX = Dialog
{
       TITLE = 'Choose a sentence'
       POSITION = 200,300
       SIZE = 266,62
       STYLE = CLOSABLE,MOVABLE
       CHILDREN = Hi, Bye, Morning, OK, Cancel
}
```

When this file is compiled, the following error messages are produced:

```
RLRC RESOURCE.UCL
RealLink for Windows Resource Compiler - Version 1.0 Rev A
(c) Copyright 1992
McDonnell Douglas Information Systems Limited

- EQUATE or EQU without corresponding TO
Line 7 - All the parameters required for create have not been set up
Line 16 - compiling continued
Line 23 - Syntax error
Line 30 - compiling continued
Line 43 - Syntax error
```

- EQUATE or EQU without corresponding TO

This is a pre-processor error caused by error ❶; the offending line has been ignored. To locate this error, examine the original source file.

Line 7	This error was caused by error ❷, but was not detected until the closing brace. The line number refers to a line in RCTEMP, which must be examined to locate the error.
Line 16	This is the line at which compilation continued after the error in line 7.

Line 23	This syntax error is caused by error ❶. Because of this error, the OK token in the **TitledButton** definition could not be changed to an identifier value.
Line 30	This is the line at which compilation continued after the error in line 23.
Line 43	This syntax error is also caused by error ❶. Once again, the OK token could not be changed to an identifier value, resulting in an invalid CHILDREN statement.

The errors in lines 7, 23 and 43 can best be found by examining RCTEMP. The temporary file produced by the above example is shown below. The lines reported by the compilation process are marked with the number of the source error concerned. The lines in the example are numbered for clarity; in a real RCTEMP file they would not be numbered, though your text editor may be able to display line numbers.

```
1
2       OPTIONBUTTON = 201
3       {
4             TITLE = 'Hello'
5       *     SIZE = 68,16
6             POSITION = 10,6
7       } ❷
8
9       OPTIONBUTTON = 202
10      {
11            TITLE = 'Good-bye'
12            POSITION = 10,23
13            SIZE = 96,16
14      }
15
16      OPTIONBUTTON = 203 ❷
17      {
18            TITLE = 'Good morning'
19            POSITION = 10,40
20            SIZE = 133,16
21      }
22
23      TITLEDBUTTON = OK ❶
24      {
25            TITLE = 'OK'
26            POSITION = 167,7
27            SIZE = 88,21
28      }
29
```

```
TITLEDBUTTON = 222 ❶
{
      TITLE = 'CANCEL'
      POSITION = 167,31
      SIZE = 88,21
}

DIALOGBOX = 200
{
      TITLE = 'Choose a sentence'
      POSITION = 200,300
      SIZE = 266,62
      STYLE = CLOSABLE,MOVABLE
      CHILDREN = 201, 202, 203, OK ❶, 222
}
```

Using the Compiled Resources

The compiled resource file must be held on the PC, in the directory specified by the `resourcepath` variable in the `[reallink]` section of the RFW.INI file; this file is held in the Windows program directory on the PC.

An application loads the resources by calling the **LoadAppRes** subroutine, specifying the handle of the application context, the name of the file containing the resources and a variable in which to return an error. For example, the following loads the resources contained in the file RESOURCE.RES:

```
CALL LoadAppRes(CONTEXT, "RESOURCE.RES", ERR)
```

Once loaded, the objects and contacts concerned can be used in the same way as those created with the create subroutines.

Chapter 8
The Help System

This chapter describes how to provide the user of a UIMS application with on-line help.

Introduction

A UIMS application provides help to the user by means of an **AppHelp** object. This consists of a compiled Windows Help Resource file on the PC that contains named sections of help text. The help text is displayed in the Windows help window, which provides search and browse facilities. In addition, sections of the help file can be linked by means of 'hot words' embedded in the text, which act as links to other sections of the file; if the user clicks on a hot word, the associated section of the help file is displayed. The help file can also contain an index, containing hot words which give access to every section of the file.

There are two ways in which the user can be given help:

- The application can display a specified section of the Help file by calling the **AppHelp** subroutine. The programmer must provide the user with access to the help file by, for instance, creating a Help menu.

- A help key can be defined which, when pressed, will display the section of the help file appropriate to the context. The programmer must link contacts displayed by the application to the corresponding sections of the help file.

Creating the Help File

The process of creating a Help Resource file is described in detail in the *Tools* manual supplied with the Windows Software Development Kit. The following summarises the requirements:

- One or more Help Topic files, saved in Microsoft Rich Text Format (RTF). Word processors that support RTF include Microsoft Word for Windows and Word for DOS.

- A Help Project file, which specifies the files which will be compiled into the application help file and various compile options. Note that UIMS can only access a section of the help file by means of a Help Index number; this means that the Help Project file must contain a [Map] section to assign a Help Index number (the Microsoft term is *context number*) to each section of the help file.

- The Help Project file and Help Topic files must be compiled using the Windows help compiler (HC), to form a Help Resource file for the application.

- The Help Resource file must be loaded onto the PC. It is recommended that it be placed in the directory specified in the RFW.INI file, or in a sub-directory of this directory.

Other development tools are available which include the Windows Help compiler and a description of how to create the various Help files. In particular, we recommend Microsoft Visual Basic 3.0 for Windows, Professional Edition.

Making Help Available to the User

The first step in making help available to the user is to load a Help Resource file by calling the **SetHelpFile** subroutine. This has the following syntax:

SetHelpFile(*Context*, *Filename*, *vErr*)

where

Context	is the handle of the application's context.
Filename	is a string containing the name of the help file. If no path name is specified, the file is loaded from the disk and directory specified in the RFW.INI file on the PC.
vErr	is a variable in which to return the completion status of the subroutine.

Once the Help Resource file has been loaded, there are two ways in which the application programmer can make help available to the user:

- By associating contacts with sections of the Help file. The user can then press the Help key to display context-sensitive help – that is, the section of the Help file which is appropriate to the command they are using.
- By providing the user with some other means of access to the Help system – the most usual is a Help menu, though some applications also provide Help buttons which display context-sensitive help.

Context-sensitive Help

Context-sensitive help using the Help key is provided as follows:

- The Help Project file must include an entry in its [Map] section, assigning a Help Index (context number) to the appropriate section of the Help file.
- The contact concerned must be associated with this section of the Help file by calling the **SetHelpIndex** subroutine. This has the following syntax:

  SetHelpIndex(*Context*, *Contact*, *Section*, *vErr*)

  where

Context	is the handle of the application's context.

Contact	is the handle of the contact for which help is being provided.
Section	is the help index (context number) of the section of the help file that is to be associated with this contact.
vErr	is a variable in which to return the completion status of the subroutine.

The Help key is normally function key F1, but can be changed if required by calling the **SetHelpKey** subroutine.

Creating a Help Menu

A help menu is created in the same way as any other menu: that is either by separately creating a menu and its menu items using **CreatePullDownMenu** and **CreateMenuItem**, or by using **MakePullDownMenu** to create the complete menu in one operation.

Within the application's message loop, **UIMS.MSG.MENUITEM** messages which originate in the help menu must initiate a call to the **AppHelp** subroutine. This has the following syntax:

AppHelp(*Context*, *Section*, *vErr*)

where

Context	The handle of the **AppContext**.
Section	The help index of the required section of the help file. If this parameter is 0, the index will be displayed.
vErr	This is a variable in which to return the completion status of the subroutine.

For example, if the Help file for your application contains a topic that describes how the keyboard is used, you could place a 'Keyboard' item on your Help menu. When the user selects that item, your application would call **AppHelp**, requesting the keyboard topic as shown below:

```
CASE CONTACT = HELP.KEYBOARD
      CALL AppHelp(CONTEXT, HELP.KEYBD.ID, ERR)
```

The **AppHelp** subroutine must also be used if you provide help buttons for the user.

Help Subroutines

The following lists all the help subroutines that are available:

SetHelpFile	Attaches a help file to the application.

GetHelpFile	Returns the name of the application's help file.
AppHelp	Displays a specified section of the help file.
SetHelpIndex	Associates a contact with a section of the help file.
GetHelpIndex	Returns the name of the help file section which is associated with a specified contact.
SetHelpKey	Assigns a key as the help accelerator.
GetHelpKey	Returns the key currently assigned as the help accelerator.

These are described in detail in Chapter 6.

Appendix A
Key Aliases

This Appendix lists the symbolic constant names, decimal values and descriptive information for the UIMS key aliases. The codes are listed in numeric order.

Table A-1. Key Aliases

UIMS Key Alias	Value	Keycap	Description
UIK.0	48	0	
UIK.1	49	1	
UIK.2	50	2	
UIK.3	51	3	
UIK.4	52	4	
UIK.5	53	5	
UIK.6	54	6	
UIK.7	55	7	
UIK.8	56	8	
UIK.9	57	9	
UIK.A	65	A	
UIK.AMPERSAND	38	&	Ampersand key
UIK.APOSTROPHE	39	'	Apostrophe (single quote) key
UIK.ASTERISK	42	*	Asterisk key
UIK.AT	64	@	At key
UIK.B	66	B	
UIK.BACKSLASH	92	\	Backslash key
UIK.BACKSPACE	8	←	Backspace key
UIK.BAR	124	|	Vertical bar key
UIK.BRACELEFT	123	{	Open curly bracket key
UIK.BRACERIGHT	125	}	Close curly bracket key
UIK.BRACKETLEFT	91	[	Open square bracket key
UIK.BRACKETRIGHT	93	]	Close square bracket key
UIK.C	67	C	
UIK.CANCEL	272		
UIK.CIRCUMFLEX	94	^	Circumflex (caret) key
UIK.CLEAR	12		
UIK.COLON	58	:	Colon key
UIK.COMMA	44	,	Comma key
UIK.D	68	D	
UIK.DELETE	127		DELETE
UIK.DOLLAR	36	$	Dollar key

(continued)

Table A-1. Key Aliases (continued)

UIMS Key Alias	Value	Keycap	Description
UIK.DOWN	257	↓	Down cursor key
UIK.E	69	E	
UIK.END	264	END	
UIK.EQUAL	61	=	Equals key
UIK.ESCAPE	27	ESC	
UIK.EXCLAM	33	!	Exclamation mark key
UIK.F	70	F	
UIK.F1	512	F1	Function key
UIK.F2	513	F2	Function key
UIK.F3	514	F3	Function key
UIK.F4	515	F4	Function key
UIK.F5	516	F5	Function key
UIK.F6	517	F6	Function key
UIK.F7	518	F7	Function key
UIK.F8	519	F8	Function key
UIK.F9	520	F9	Function key
UIK.F10	521	F10	Function key
UIK.F11	522	F11	Function key
UIK.F12	523	F12	Function key
UIK.F13	524	F13	Function key
UIK.F14	525	F14	Function key
UIK.F15	526	F15	Function key
UIK.G	71	G	
UIK.GRAVE	96	`	Open single quote key
UIK.GREATER	62	>	Greater than key
UIK.H	72	H	
UIK.HELP	265		
UIK.HOME	263	HOME	
UIK.I	73	I	
UIK.INSERT	262	INSERT	

(continued)

Table A-1. Key Aliases (continued)

UIMS Key Alias	Value	Keycap	Description
UIK.J	74	J	
UIK.K	75	K	
UIK.L	76	L	
UIK.LEFT	258	←	Left cursor key
UIK.LESS	60	<	Less than key
UIK.M	77	M	
UIK.MINUS	45	-	Minus key
UIK.MULTI00	128		
UIK.MULTI01	129		
UIK.MULTI02	130		
UIK.MULTI03	131		
UIK.MULTI04	132		
UIK.MULTI05	133		
UIK.MULTI06	134		
UIK.MULTI07	135		
UIK.MULTI08	136		
UIK.MULTI09	137		
UIK.MULTI0A	138		
UIK.MULTI0B	139		
UIK.MULTI0C	140		
UIK.MULTI0D	141		
UIK.MULTI0E	142		
UIK.MULTI0F	143		
UIK.MULTI10	144		
UIK.MULTI10	144		
UIK.MULTI11	145	‘	open single quote
UIK.MULTI12	146	’	close single quote
UIK.MULTI13	147		
UIK.MULTI14	148		
UIK.MULTI15	149		
UIK.MULTI16	150		
UIK.MULTI17	151		
UIK.MULTI18	152		

(continued)

Table A-1. Key Aliases (continued)

UIMS Key Alias	Value	Keycap	Description
UIK.MULTI19	153		
UIK.MULTI1A	154		
UIK.MULTI1B	155		
UIK.MULTI1C	156		
UIK.MULTI1D	157		
UIK.MULTI1E	158		
UIK.MULTI1F	159		
UIK.MULTI20	160		space
UIK.MULTI21	161	¡	
UIK.MULTI22	162	¢	
UIK.MULTI23	163	£	
UIK.MULTI24	164	¤	
UIK.MULTI25	165	¥	
UIK.MULTI26	166	¦	
UIK.MULTI27	167	§	
UIK.MULTI28	168	¨	
UIK.MULTI29	169	©	
UIK.MULTI2A	170	ª	
UIK.MULTI2B	171	«	
UIK.MULTI2C	172	¬	
UIK.MULTI2D	173	-	
UIK.MULTI2E	174	®	
UIK.MULTI2F	175	¯	
UIK.MULTI30	176	°	
UIK.MULTI31	177	±	
UIK.MULTI32	178	²	
UIK.MULTI33	179	³	
UIK.MULTI34	180	´	
UIK.MULTI35	181	µ	
UIK.MULTI36	182	¶	
UIK.MULTI37	183	·	
UIK.MULTI38	184	¸	
UIK.MULTI39	185	¹	
UIK.MULTI3A	186	º	
UIK.MULTI3B	187	»	
UIK.MULTI3C	188	¼	

(continued)

Table A-1. **Key Aliases (continued)**

UIMS Key Alias	Value	Keycap	Description
UIK.MULTI3D	189	½	
UIK.MULTI3E	190	¾	
UIK.MULTI3F	191	¿	
UIK.MULTI40	192	À	
UIK.MULTI41	193	Á	
UIK.MULTI42	194	Â	
UIK.MULTI43	195	Ã	
UIK.MULTI44	196	Ä	
UIK.MULTI45	197	Å	
UIK.MULTI46	198	Æ	
UIK.MULTI47	199	Ç	
UIK.MULTI48	200	È	
UIK.MULTI49	201	É	
UIK.MULTI4A	202	Ê	
UIK.MULTI4B	203	Ë	
UIK.MULTI4C	204	Ì	
UIK.MULTI4D	205	Í	
UIK.MULTI4E	206	Î	
UIK.MULTI4F	207	Ï	
UIK.MULTI50	208	Ð	
UIK.MULTI51	209	Ñ	
UIK.MULTI52	210	Ò	
UIK.MULTI53	211	Ó	
UIK.MULTI54	212	Ô	
UIK.MULTI55	213	Õ	
UIK.MULTI56	214	Ö	
UIK.MULTI57	215	×	
UIK.MULTI58	216	Ø	
UIK.MULTI59	217	Ù	
UIK.MULTI5A	218	Ú	
UIK.MULTI5B	219	Û	
UIK.MULTI5C	220	Ü	
UIK.MULTI5D	221	Ý	
UIK.MULTI5E	222	Þ	
UIK.MULTI5F	223	ß	
UIK.MULTI60	224	à	

(continued)

Table A-1. Key Aliases (continued)

UIMS Key Alias	Value	Keycap	Description
UIK.MULTI61	225	á	
UIK.MULTI62	226	â	
UIK.MULTI63	227	ã	
UIK.MULTI64	228	ä	
UIK.MULTI65	229	å	
UIK.MULTI66	230	æ	
UIK.MULTI67	231	ç	
UIK.MULTI68	232	è	
UIK.MULTI69	233	é	
UIK.MULTI6A	234	ê	
UIK.MULTI6B	235	ë	
UIK.MULTI6C	236	ì	
UIK.MULTI6D	237	í	
UIK.MULTI6E	238	î	
UIK.MULTI6F	239	ï	
UIK.MULTI70	240	ð	
UIK.MULTI71	241	ñ	
UIK.MULTI72	242	ò	
UIK.MULTI73	243	ó	
UIK.MULTI74	244	ô	
UIK.MULTI75	245	õ	
UIK.MULTI76	246	ö	
UIK.MULTI77	247	÷	
UIK.MULTI78	248	ø	
UIK.MULTI79	249	ù	
UIK.MULTI7A	250	ú	
UIK.MULTI7B	251	û	
UIK.MULTI7C	252	ü	
UIK.MULTI7D	253	ý	
UIK.MULTI7E	254	þ	
UIK.MULTI7F	255	ÿ	
UIK.N	78	N	
UIK.NEXT	261	PAGE DOWN	
UIK.NUMBERSIGN	35	#	Number-sign key

(continued)

Table A-1. Key Aliases (continued)

UIMS Key Alias	Value	Keycap	Description
UIK.O	79	O	
UIK.P	80	P	
UIK.PARENLEFT	40	(	Open parenthesis key
UIK.PARENRIGHT	41	)	Close parenthesis key
UIK.PERCENT	37	%	Percent key
UIK.PERIOD	46	.	Period key
UIK.PLUS	43	+	Plus key
UIK.PRIOR	260	PAGE UP	
UIK.Q	81	Q	
UIK.QUESTION	63	?	Question mark key
UIK.QUOTEDBL	34	"	Double quote key
UIK.R	82	R	
UIK.RETURN	13	↵	Return key
UIK.RIGHT	259	→	Right cursor key
UIK.S	83	S	
UIK.SCRLOCK	266	SCROLL LOCK	
UIK.SEMICOLON	59	;	Semicolon key
UIK.SLASH	47	/	Slash key
UIK.SPACE	32		SPACEBAR
UIK.T	84	T	
UIK.TAB	9		TAB
UIK.TILDE	126	~	Tilde key
UIK.U	85	U	
UIK.UNDERSCORE	95	_	Underscore key
UIK.UNKNOWN	65535		Unrecognised key.
UIK.UP	256	↑	Up cursor key
UIK.V	86	V	
UIK.W	87	W	

(continued)

Table A-1. Key Aliases (continued)

UIMS Key Alias	Value	Keycap	Description
UIK.X	88	X	
UIK.Y	89	Y	
UIK.Z	90	Z	

Note: The codes **UIK.MULTI00** to **UIK.MULTI7F** are for keys specific to particular national keyboards. The keycap given for each is the standard ANSI code for the character concerned. On keyboards that do not include these keys, the codes can be generated by holding down the ALT key while entering a zero followed by the ANSI code on the numeric keypad. The code will be generated when the ALT key is released. Note, however, that with the NUMLOCK off, keypress messages will be generated as each key is operated.

Table A-2. Key Modifiers

UIMS Key Modifier	Value	Keycap	Description
UIK.CAPSLOCK	65536	CAPS LOCK	
UIK.NUMLOCK	131072	NUM LOCK	
UIK.SHIFT	262144	⇧	SHIFT
UIK.CTRL	524288	CTRL	
UIK.ALT	1048576	ALT	
UIK.NUMPAD	2097152		The key operated is on the numeric keypad.

Table A-3. Pointer Modifiers

UIMS Pointer Modifier	Value	Description
UIK.P.DRAG	2147483648	The pointer is being dragged (drag start).
UIK.P.BUTTON1	1073741824	Pointer button 1 is pressed.
UIK.P.BUTTON2	536870912	Pointer button 2 is pressed.
UIK.P.BUTTON3	268435456	Pointer button 3 is pressed.
UIK.P.BUTTON4	134217728	Pointer button 4 is pressed.
UIK.P.BUTTON5	67108864	Pointer button 5 is pressed.

Note: The pointer button combinations which produce these values are hardware dependent.

Appendix B
Screen Colours

This appendix describes how screen colours are specified in a UIMS application and lists the pre-defined logical colours. It also explains the effects of the different graphics drawing modes.

Specifying Colours

In a UIMS application, screen colours can be specified in two ways:

- The absolute colour can be specified as a particular combination of red, green and blue elements. The intensity of each of the these elements is in turn specified as an integer between 0 and 255, where 0 is zero intensity and 255 full brightness. The required combination is then created as follows:

  65536**red* + 256**green* + *blue*

  The intensities of the red, green and blue elements of a colour can be obtained as follows:

  blue = MOD(*colour*, 256)
  green = MOD(INT(*colour*/256), 256)
  red = MOD(INT(*colour*/65536), 256)

- Any one of the sixteen pre-defined logical colours listed in Table B-1 can be used.

Table B-1. Logical Colour Bindings

Logical Colour	Red	Green	Blue
UIMS.BLACK	0	0	0
UIMS.BLUE	0	0	255
UIMS.BROWN	128	128	0
UIMS.CYAN	0	255	255
UIMS.DARKBLUE	0	0	128
UIMS.DARKCYAN	0	128	128
UIMS.DARKGREEN	0	128	0
UIMS.DARKGREY	85	85	85
UIMS.DARKMAGENTA	128	0	128
UIMS.DARKRED	128	0	0
UIMS.GREEN	0	255	0

(continued)

Table B-1 Logical Colour Bindings (continued)

Logical Colour	Red	Green	Blue
UIMS.GREY	170	170	170
UIMS.MAGENTA	255	0	255
UIMS.RED	255	0	0
UIMS.WHITE	255	255	255
UIMS.YELLOW	255	255	0

Graphics Drawing Modes

The appearance of lines drawn on the display is determined not only by the colour of the **Pen** object, but also by the graphics drawing mode selected in the **Drawrule**. Eight modes are available:

UIMS.DRAW.CLEAR
For each pixel, a new colour is produced by inverting the pen colour bit-wise, and then performing a bit-wise AND between the result and the current colour of the destination pixel.

UIMS.DRAW.COPY
Lines are drawn in the pen colour, regardless of the colour of the destination.

UIMS.DRAW.NOTCLEAR
For each pixel, a new colour is produced by performing a bit-wise AND between the pen colour and the current colour of the destination pixel.

UIMS.DRAW.NOTCOPY
Lines are drawn in the bit-wise inverse of the pen colour, regardless of the colour of the destination.

UIMS.DRAW.NOTOR
For each pixel, a new colour is produced by inverting the pen colour, and then performing a bit-wise OR between the result and the current colour of the destination pixel.

UIMS.DRAW.NOTXOR
For each pixel, a new colour is produced by performing a bit-wise exclusive-OR between the pen colour and the current colour of the destination pixel, and then inverting the result.

UIMS.DRAW.OR
For each pixel, a new colour is produced by performing a bit-wise OR between the pen colour and the current colour of the destination pixel.

UIMS.DRAW.XOR
For each pixel, a new colour is produced by performing a bit-wise exclusive-OR between the pen colour and the current colour of the destination pixel.

The different drawing modes are best understood by considering what happens when two lines, one black and one white, are drawn across a screen which is part white and part black. The results are summarised below:

Screen	Pen	CLEAR	COPY	NOTCLEAR	NOTCOPY	NOTOR	NOTXOR	OR	XOR
White	White	White	White	White	Black	Black	Black	White	White
White	Black	White	Black	White	White	White	White	Black	Black
Black	White	Black	White	White	Black	Black	White	Black	Black
Black	Black	White	Black	Black	White	Black	Black	Black	White

This is, of course, the simplest case. In reality, even on a monochrome display UIMS can produce various shades of grey by dithering black and white pixels. Since the logical operations are carried out on a pixel-by-pixel basis, the result of drawing a pure white or black line on a grey background will in most cases be a different shade of grey

The situation becomes even more complex on a colour display, since each of the three primary colours (red, green and blue) is affected separately by the logical operation. This can be illustrated by considering the **UIMS.DRAW.NOTCOPY** drawing mode, which simply inverts the pen colour and replaces the screen colour with the result.

Pen Colour	Result
Black	White
Blue	Yellow
Green	Magenta
Cyan	Red
Red	Cyan
Magenta	Green
Yellow	Blue
White	Black

The following tables show the resulting colours for all combinations of red, green and blue in the Pen colour and destination pixel, for the remaining drawing modes.

Table B-2. UIMS.DRAW.CLEAR Colour Combinations

	Pen Colour							
Destination	**Black**	**Blue**	**Green**	**Cyan**	**Red**	**Magenta**	**Yellow**	**White**
Black	White	Yellow	Magenta	Red	Cyan	Green	Blue	Black
Blue	White	White	Magenta	Magenta	Cyan	Cyan	Blue	Blue
Green	White	Yellow	White	Yellow	Cyan	Green	Cyan	Green
Cyan	White	White	White	White	Cyan	Cyan	Cyan	Cyan
Red	White	Yellow	Magenta	Red	White	Yellow	Magenta	Red
Magenta	White	White	Magenta	Magenta	White	White	Magenta	Magenta
Yellow	White	Yellow	White	Yellow	White	Yellow	White	Yellow
White	White	White	White	White	White	White	White	White

Table B-3. UIMS.DRAW.NOTCLEAR Colour Combinations

	Pen Colour							
Destination	**Black**	**Blue**	**Green**	**Cyan**	**Red**	**Magenta**	**Yellow**	**White**
Black	Black	Blue	Green	Cyan	Red	Magenta	Yellow	White
Blue	Blue	Blue	Cyan	Cyan	Magenta	Magenta	White	White
Green	Green	Cyan	Green	Cyan	Yellow	White	Yellow	White
Cyan	Cyan	Cyan	Cyan	Cyan	White	White	White	White
Red	Red	Magenta	Yellow	White	Red	Magenta	Yellow	White
Magenta	Magenta	Magenta	White	White	Magenta	Magenta	White	White
Yellow	Yellow	White	Yellow	White	Yellow	White	Yellow	White
White	White	White	White	White	White	White	White	White

Table B-4. UIMS.DRAW.NOTOR Colour Combinations

	Pen Colour							
Destination	**Black**	**Blue**	**Green**	**Cyan**	**Red**	**Magenta**	**Yellow**	**White**
Black	Black	Black	Black	Black	Black	Black	Black	Black
Blue	Blue	Black	Blue	Black	Blue	Black	Blue	Black
Green	Green	Green	Black	Black	Green	Green	Black	Black
Cyan	Cyan	Green	Blue	Black	Cyan	Green	Blue	Black
Red	Red	Red	Red	Red	Black	Black	Black	Black
Magenta	Magenta	Red	Magenta	Red	Blue	Black	Blue	Black
Yellow	Yellow	Yellow	Red	Red	Green	Green	Black	Black
White	White	Yellow	Magenta	Red	Cyan	Green	Blue	Black

Table B-5. UIMS.DRAW.NOTXOR Colour Combinations

	Pen Colour							
Destination	**Black**	**Blue**	**Green**	**Cyan**	**Red**	**Magenta**	**Yellow**	**White**
Black	Black	Blue	Green	Cyan	Red	Magenta	Yellow	White
Blue	Blue	Black	Cyan	Green	Magenta	Red	White	Yellow
Green	Green	Cyan	Black	Blue	Yellow	White	Red	Magenta
Cyan	Cyan	Green	Blue	Black	White	Yellow	Magenta	Red
Red	Red	Magenta	Yellow	White	Black	Blue	Green	Cyan
Magenta	Magenta	Red	White	Yellow	Blue	Black	Cyan	Green
Yellow	Yellow	White	Red	Magenta	Green	Cyan	Black	Blue
White	White	Yellow	Magenta	Red	Cyan	Green	Blue	Black

Table B-6. UIMS.DRAW.OR Colour Combinations

	Pen Colour							
Destination	**Black**	**Blue**	**Green**	**Cyan**	**Red**	**Magenta**	**Yellow**	**White**
Black	Black	Black	Black	Black	Black	Black	Black	Black
Blue	Black	Blue	Black	Blue	Black	Blue	Black	Blue
Green	Black	Black	Green	Green	Black	Black	Green	Green
Cyan	Black	Blue	Green	Cyan	Black	Blue	Green	Cyan
Red	Black	Black	Black	Black	Red	Red	Red	Red
Magenta	Black	Blue	Black	Blue	Red	Magenta	Red	Magenta
Yellow	Black	Black	Green	Green	Red	Red	Yellow	Yellow
White	Black	Blue	Green	Cyan	Red	Magenta	Yellow	White

Table B-7. UIMS.DRAW.XOR Colour Combinations

	Pen Colour							
Destination	**Black**	**Blue**	**Green**	**Cyan**	**Red**	**Magenta**	**Yellow**	**White**
Black	White	Yellow	Magenta	Red	Cyan	Green	Blue	Black
Blue	Yellow	White	Red	Magenta	Green	Cyan	Black	Blue
Green	Magenta	Red	White	Yellow	Blue	Black	Cyan	Green
Cyan	Red	Magenta	Yellow	White	Black	Blue	Green	Cyan
Red	Cyan	Green	Blue	Black	White	Yellow	Magenta	Red
Magenta	Green	Cyan	Black	Blue	Yellow	White	Red	Magenta
Yellow	Blue	Black	Cyan	Green	Magenta	Red	White	Yellow
White	Black	Blue	Green	Cyan	Red	Magenta	Yellow	White

Appendix C
Resource Compiler Keywords

This appendix lists the object type and attribute keywords recognised by the resource compiler and gives details of mandatory attributes and valid attribute settings. It also lists the error messages that might be displayed by the resource compiler and suggests probable causes for these.

Object Types

APPWINDOW	BRUSH	CHECKBUTTON
CHILDWINDOW	DIALOGBOX	DRAWRULE
EDITBOX	EXCLUSIVEGRP	INCLUSIVEGRP
LINE	LISTBOX	MENU
MENUBAR	MENUITEM	OPTIONBUTTON
PEN	POINTER	RECTANGLE
SCROLLBAR	TEXT	TEXTEDITOR
TITLEDBUTTON		

Object Attributes

This section lists the attributes which are valid for each type of object.

Note: Attributes in **bold** are mandatory; they must be included every time an object of the specified type is defined.

APPWINDOW	**BDRSTYLE**	Border style. See page C-11 for valid settings.
	CHILDREN	List of Object IDs.
	CLIPREGION	List of four coordinate values (top, left, bottom, right).
	CURSORPOS	Cursor position - list of two coordinate values (horizontal, vertical).
	CURSORSTATE	List of two settings (Visible, Type). Visible - one of: TRUE FALSE Type - one of: OUTLINE BLOCK UNDERLINE BAR
	DRAWRULE	Object ID.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	MENUBAR	Object ID.
	POINTER	Object ID.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	SIZING	One of: MAX MIN NORMAL
	STYLE	List of settings, each one of: CLOSABLE DIALOG HSCROLL ICONISABLE MOVABLE NONE SIZABLE

		TEXT VSCROLL
	TITLE	String.
	UPDATE	See page C-11 for valid settings.
BRUSH	**FOREGROUND**	See page C-11 for valid settings.
	STYLE	One of: HOLLOW SOLID
CHECKBUTTON	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SELECTED	One of: TRUE FALSE
	SIZE	List of two coordinate values (width, height).
	TITLE	String.
	TOGGLE	One of: TRUE FALSE
	UPDATE	See page C-11 for valid settings.
CHILDWINDOW	**BDRSTYLE**	Border style. See page C-11 for valid settings.
	CHILDREN	List of Object IDs.
	CLIPREGION	List of four coordinate values (top, left, bottom, right).
	CURSORPOS	Cursor position - list of two coordinate values (horizontal, vertical).
	CURSORSTATE	List of two settings (Visible, Type). Visible - one of: TRUE FALSE Type - one of: OUTLINE BLOCK UNDERLINE BAR
	DRAWRULE	Object ID.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POINTER	Object ID.
	POSITION	List of two coordinate values (horizontal, vertical).

	SIZE	List of two coordinate values (width, height).
	STYLE	List of settings, each one of:
		DIALOG
		HSCROLL
		NONE
		TEXT
		VSCROLL
	UPDATE	See page C-11 for valid settings.
DIALOGBOX	CHILDREN	List of Object IDs.
	DEFBUTTON	Object ID.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	MODE	One of: APP
		LESS
		SYS
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	STYLE	List of settings, each one of:
		CLOSABLE
		MOVABLE
		NONE
	TITLE	String.
	UPDATE	See page C-11 for valid settings.
DRAWRULE	**BACKGROUND**	See page C-11 for valid settings.
	BRUSH	Object ID.
	DRAWMODE	One of: CLEAR
		COPY
		NOTCLEAR
		NOTCOPY
		NOTOR
		NOTXOR
		OR
		XOR
	FOREGROUND	See page C-11 for valid settings.
	PEN	Object ID.
	TEXTMODE	One of: OPAQUE
		HOLLOW

EDITBOX.	CONTENT	String.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	MASK	This parameter is for future use. It must be set to a null string when defining an **EditBox**, but its value will be ignored when the object is created.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	STYLE	List of settings, each one of: BORDER NONE
	UPDATE	See page C-11 for valid settings.
EXCLUSIVEGRP	**BORDER**	Border style. See page C-11 for valid settings.
	CHILDREN	List of Object IDs.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	TITLE	String.
	UPDATE	See page C-11 for valid settings.
INCLUSIVEGRP	**BORDER**	Border style. See page C-11 for valid settings.
	CHILDREN	List of Object IDs.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	TITLE	String.
	UPDATE	See page C-11 for valid settings.
LINE	DRAWRULE	Object ID.
	ENABLED	See page C-11 for valid settings.
	ENDPOS	List of two coordinate values (horizontal, vertical). Note that the position must be specified relative to STARTPOS.

	ENDSTYLE	This parameter is for future use. It must be set to DEFAULT when defining a **Line** contact, but its value will be ignored when the object is created.
	MAPPED	See page C-11 for valid settings.
	STARTPOS	List of two coordinate values (horizontal, vertical).
	UPDATE	See page C-11 for valid settings.
LISTBOX	CONTENT	List of strings.
	CONTROLS	One of: NONE MULTISELECT
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	LINK	Object ID.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SELECTION	Value.
	SIZE	List of two coordinate values (width, height).
	UPDATE	See page C-11 for valid settings.
MENU	CHILDREN	List of Object IDs.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	TITLE	String.
MENUBAR	CHILDREN	List of Object IDs.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	MAPPED	See page C-11 for valid settings.
	UPDATE	See page C-11 for valid settings.
MENUITEM	AUTOCHECK	One of: TRUE FALSE
	CHECKMARK	One of: TRUE FALSE
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	TITLE	String.

OPTIONBUTTON	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SELECTED	One of: TRUE FALSE
	SIZE	List of two coordinate values (width, height).
	TITLE	String.
	TOGGLE	One of: TRUE FALSE
	UPDATE	See page C-11 for valid settings.
PEN	**FOREGROUND**	See page C-11 for valid settings.
	STYLE	One of: HOLLOW SOLID
	WIDTH	Number of pixels.
POINTER	**TYPE**	One of: ARROW CROSS CUSTOM IBEAM PLUS WAIT
RECTANGLE	DRAWRULE	Object ID.
	ENABLED	See page C-11 for valid settings.
	ENDPOS	List of two coordinate values (horizontal, vertical). Note that the position must be specified relative to STARTPOS.
	MAPPED	See page C-11 for valid settings.
	STARTPOS	List of two coordinate values (horizontal, vertical).
	STYLE	One of: NONE BORDER
	UPDATE	See page C-11 for valid settings.
SCROLLBAR	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	INC	List of two increment values (page, line).
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	RANGE	List of two coordinate values (minimum, maximum).
	SIZE	List of two coordinate values (width, height).

	THUMBPOS	Value.
	TRACK	One of: TRUE FALSE
	TYPE	One of: HORZ VERT
	UPDATE	See page C-11 for valid settings.
TEXT	**CONTENT**	String.
	DRAWRULE	Object ID.
	ENABLED	See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	JUSTIFICATION	One of: BOTH CENTRED LEFT RIGHT
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	UPDATE	See page C-11 for valid settings.
TEXTEDITOR	CONTENT	String.
	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).
	STYLE	List of settings, each one of: NONE AUTOSCROLL BORDER HSCROLLBAR READONLY VSCROLLBAR
	UPDATE	See page C-11 for valid settings.
TITLEDBUTTON	ENABLED	See page C-11 for valid settings.
	EVENTMASK	List of settings. See page C-11 for valid settings.
	HELPINDEX	Help identifier value.
	MAPPED	See page C-11 for valid settings.
	POSITION	List of two coordinate values (horizontal, vertical).
	SIZE	List of two coordinate values (width, height).

STYLE	List of settings, each one of: BORDER NONE THICK TRANS
TITLE	String.
UPDATE	See page C-11 for valid settings.

Common Object Attributes

This section lists attributes which are common to a number objects and contacts, or which have a large number of possible settings.

Border Styles	One of:	NONE		
		BORDER		
ENABLED	One of:	TRUE		
		FALSE		
EVENTMASK	List of:	BUTTONPRESS	CLICK	CLOSE
		DBLCLICK	DRAG	ENTER
		EXIT	HSCROLL	IDLE
		KEYPRESS	KILL	LBOXDESELECT
		LBOXSELECT	LEAVE	MENUITEM
		MOTION	MOVE	NEWVIEW
		NOTIFY	PRESS	RELEASE
		SCROLL	SELECT	SIZE
		TIMER	UPDATE	VSCROLL

MAPPED Whether or not the contact is to be visible on the screen. One of:
TRUE
FALSE

UPDATE Update mode. One of:
NONE
IMMEDIATE

Colours	One of:	BLACK	BLUE	BROWN
		CYAN	DARKBLUE	DARKCYAN
		DARKGREEN	DARKGREY	DARKMAGENTA
		DARKRED	GREEN	GREY
		MAGENTA	RED	WHITE
		YELLOW		

Note: In a Resource Script you can only specify logical colours - you cannot define colours as combinations of red, green and blue.

Virtual Keys	One of:	0	1	2
		3	4	5
		6	7	8
		9	A	AMPERSAND

APOSTROPHE	ASTERISK	AT
B	BACKSLASH	BACKSPACE
BAR	BRACELEFT	BRACERIGHT
BRACKETLEFT	BRACKETRIGHT	C
CIRCUMFLEX	CLEAR	COLON
COMMA	D	DELETE
DOLLAR	DOWN	E
END	EQUAL	ESCAPE
EXCLAM	F	F1
F10	F11	F12
F13	F14	F15
F2	F3	F4
F5	F6	F7
F8	F9	G
GREATER	H	HELP
HOME	I	INSERT
J	K	L
LEFT	LESS	M
MINUS	MULTI00	MULTI01
MULTI02	MULTI03	MULTI04
MULTI05	MULTI06	MULTI07
MULTI08	MULTI09	MULTI0A
MULTI0B	MULTI0C	MULTI0D
MULTI0E	MULTI0F	MULTI10
MULTI11	MULTI12	MULTI13
MULTI14	MULTI15	MULTI16
MULTI17	MULTI18	MULTI19
MULTI1A	MULTI1B	MULTI1C
MULTI1D	MULTI1E	MULTI1F
MULTI20	MULTI21	MULTI22
MULTI23	MULTI24	MULTI25
MULTI26	MULTI27	MULTI28
MULTI29	MULTI2A	MULTI2B
MULTI2C	MULTI2D	MULTI2E
MULTI2F	MULTI30	MULTI31
MULTI32	MULTI33	MULTI34
MULTI35	MULTI36	MULTI37
MULTI38	MULTI39	MULTI3A
MULTI3B	MULTI3C	MULTI3D
MULTI3E	MULTI3F	MULTI40
MULTI41	MULTI42	MULTI43
MULTI44	MULTI45	MULTI46
MULTI47	MULTI48	MULTI49

MULTI4A	MULTI4B	MULTI4C
MULTI4D	MULTI4E	MULTI4F
MULTI50	MULTI51	MULTI52
MULTI53	MULTI54	MULTI55
MULTI56	MULTI57	MULTI58
MULTI59	MULTI5A	MULTI5B
MULTI5C	MULTI5D	MULTI5E
MULTI5F	MULTI60	MULTI61
MULTI62	MULTI63	MULTI64
MULTI65	MULTI66	MULTI67
MULTI68	MULTI69	MULTI6A
MULTI6B	MULTI6C	MULTI6D
MULTI6E	MULTI6F	MULTI70
MULTI71	MULTI72	MULTI73
MULTI74	MULTI75	MULTI76
MULTI77	MULTI78	MULTI79
MULTI7A	MULTI7B	MULTI7C
MULTI7D	MULTI7E	MULTI7F
N	NEXT	NUMBERSIGN
O	P	PARENLEFT
PARENRIGHT	PERCENT	PERIOD
PLUS	PRIOR	Q
QUOTERIGHT	QUESTION	QUOTEDBL
QUOTELEFT	R	RETURN
RIGHT	S	SEMICOLON
SLASH	SPACE	T
TAB	TILDE	U
UNDERSCORE	UP	V
W	X	Y
Z		

Key Modifiers

Any of the above virtual key codes can be combined with one or more of the following key modifiers. The keys must be separated by plus (+) signs; for example: `CTRL+F`, `CTRL+SHIFT+F4`.

CAPSLOCK
NUMLOCK
SHIFT
CTRL
ALT

Errors

This section lists the error messages which might be displayed during pre-processing and compilation. In each case, the meaning is explained and appropriate action suggested.

Command Errors

These errors can occur when you type in the RLRC command, but before pre-processing or compiling starts.

`Can't open file` *filename*

RLRC cannot find the resource file you have specified. Enter the correct file name.

`Can't open message file - rc.msg`

The resource compiler's message file cannot be found. Possible causes are:

- The resource compiler has been copied to a different directory. Copy the files RC.MSG and RC.DAT as well as RLRC.EXE.
- A disk error has occurred. Use a disk maintenance tool to find and correct the error and then re-install the resource compiler from your RealLink for Windows disks.

`Can't open temporary file - 'rctemp'`

The resource compiler is unable to create the temporary, pre-processed file. Possible causes are:

- The disk you are using is write protected. Enable writing to the disk or use a different disk.
- The disk you are using is full. Use a different disk, or delete unwanted files to create more space.
- The directory in which you are compiling contains too many files; this can normally only occur in the root (\) directory. Change to a different directory.

`Resource compiler needs '.ucl' or '.UCL' suffix`

Your resource file has the wrong file extension. Rename your file.

`Resource script filename (.ucl) :`

You have omitted the name of the file containing the resource script in your RLRC command line. Enter the name of the required file.

Pre-processor Errors

If an error occurs during pre-processing, an error message is displayed and the line containing the error is ignored.

`Can't open data file - rc.dat`

The resource compiler's data file cannot be found. Possible causes are:

- The resource compiler has been copied to a different directory. Copy the files RC.MSG and RC.DAT as well as RLRC.EXE.

- A disk error has occurred. Use a disk maintenance tool to find and correct the error and then re-install the resource compiler from your RealLink for Windows disks.

`- #ELSE without corresponding #IFDEF`

The #ELSE pre-processor command is only legal if preceded by an #IFDEF command. Check the structure of your source file.

`- #ENDIF without corresponding #IFDEF`

The #ENDIF pre-processor command is only legal if preceded by an #IFDEF command. Check the structure of your source file.

`- EQUATE or EQU without corresponding TO`

The TO keyword and/or the value has been omitted from an EQUATE or EQU pre-processor statement. Check your resource script and any included files.

filename `is not valid`

The file specified in an #INCLUDE statement cannot be found. Check that the file name is spelled correctly and that the file concerned is accessible to the resource compiler.

`- Include file must have '.ucl', '.UCL', '.h' or '.H' suffix`

You have specified a file to be included which has an illegal suffix. Check your resource script and any included files.

`Line` *number* `- More than 5 levels of #INCLUDE, ignored`

Included files have been nested too deeply. Reorganise your source files.

`- More than 9 levels of #IFDEF, ignored`

An #IFDEF structure has been nested too deeply. Reorganise the structure of your source file.

Compilation Errors

If an error occurs during compilation, the number of the line in which the error occurred is displayed, together with an error message. All subsequent source lines are ignored, up to the closing brace of the current outer nested level. Compilation then continues from this point.

Note that the line numbers reported are not those in the original source file, but in a temporary file, RCTEMP, created in the current directory.

`Line` *number* `- All the parameters required for create have not been set up`

You have omitted one or more mandatory parameters in an object definition. Check the RCTEMP temporary file and correct your resource script.

`Line` *number* `- compiling continued`

After an error, compiling has continued from the specified line. The lines between that containing the error and this line have not been compiled.

`Line` *number* `- Id given is within forbidden limits -` *ident*

An identifier value you have chosen is one of those reserved for internal use by UIMS. Use a value outside the range 8000 – 9999.

`Line` *number* `- Invalid UIMS type parameter -` *value*

An object attribute has been set to an invalid keyword value. Check the RCTEMP temporary file and correct your resource script. Refer to pages C-3 to C-10 for details of the valid attribute settings for each type of object.

`Line` *number* `- Object does not have property -` *attribute*

An attribute set in an object definition is not valid for the object concerned. Check the RCTEMP temporary file and correct your resource script. Refer to pages C-3 to C-10 for details of the attributes which are valid for each type of object.

Note: This error can also occur if you have used a token as the identifier for a nested object, but have not defined a value for the token.

`Line` *number* `- Object not defined -` *name*

An invalid object type has been specified. Check the RCTEMP temporary file and correct your resource script. Refer to page C-2 for the list of valid object types.

`Line` *number* `- Parameter should be a number -` *value*

You have used a string or keyword value instead of a number when setting an attribute. Check the RCTEMP temporary file and correct your resource script. Refer to pages C-3 to C-10 for details of the valid attribute settings for each type of object.

`Line` *number* `- Parameter should be a string -` *value*

You have used a number or keyword value instead of a string when setting an attribute. Check the RCTEMP temporary file and correct your resource script. Refer to pages C-3 to C-10 for details of the valid attribute settings for each type of object.

`Line` *number* `- Syntax error`

Several conditions can cause this error. The most common cause is a mis-typed resource compiler keyword. Check the RCTEMP temporary file and correct your resource script.

`Line` *number* `- Text string invalid in CHILDREN other than for MENU,MENUBAR`

An automatic **MenuItem** definition has been used in the CHILDREN attribute of an object other than a **Menu** or **MenuBar**. Check the RCTEMP temporary file and correct your resource script.

`Line` *number* `- Too few parameters`

You have supplied too few parameters when setting the value of an attribute. Check the RCTEMP temporary file and correct your resource script. Refer to pages C-3 to C-10 for details of the valid attribute settings for each type of object.

`Line` *number* `- Too many parameters`

You have supplied too many parameters when setting the value of an attribute. Check the RCTEMP temporary file and correct your resource script. Refer to pages C-3 to C-10 for details of the valid attribute settings for each type of object.

`Line` *number* `- Unpaired quote`

In defining a string value, you have omitted the closing single quote. Check the RCTEMP temporary file and correct your resource script.

Appendix D
Error Codes

This appendix lists the completion/error codes which may be returned by UIMS subroutines in the *vErr* parameter. The numeric value of each is given, together with the message that is returned for that code by the **GetErrorText** subroutine. Possible causes of each error are also suggested.

The codes listed are defined in items in the file UIMS-TOOLS; the appropriate items should be included in your application, depending on the subroutines used – see Chapter 2 for details.

UIMS Error Codes

These codes are defined in the item UIMSDEFS in the file UIMS-TOOLS.

0 **ERR.SUCCESS** `No Error`

Subroutine completed successfully.

1 **ERR.FAIL** `General failure`

A subroutine has failed, but the reason is unknown. This is usually caused by insufficient memory or Windows resources on the PC. Close down as many applications as possible and try again. If this fails, try restarting Windows.

2 **ERR.UNSUPPORTED** `Unsupported facility`

You have attempted to set a common contact attribute that does not apply to the specified contact. Refer to the contact description in Chapter 3.

3 **ERR.INVHANDLE** `Invalid handle`

The handle you have specified does not identify an object that currently exists. This might be caused by the following:

- You have used an incorrect application context handle or have failed to sign on before calling a subroutine that requires the handle of the application context.
- You have used an incorrect object handle, or that of an object that has not yet been created, or has been destroyed.

5 **ERR.MALLOC** `No memory allocated`

UIMS was unable to allocate the memory required for an operation. This is usually caused by insufficient memory or Windows resources on the PC. Close down as many applications as possible and try again. If this fails, try restarting Windows.

8 **ERR.INVCLASS** `Invalid class`

You have called a subroutine that attaches one object to another (for example: **AppWinSetMenuBar, DrawruleSetFont** or **SetPointer**) and have specified the wrong type of object. For example, passing the handle of a **Brush** object to **DrawruleSetPen** instead of that of a **Pen** will produce this error.

10 ERR.NOITEM `Couldn't find the item to delete`

- You have called **ListBoxRemoveContent** or **ListBoxRemoveContents** and have specified an item that does not exist.
- You have called **ListBoxGetSelections**, but none of the items in the list box are selected.

11 ERR.INVPARAM `Invalid parameter`

You have passed an invalid parameter to a subroutine.

12 ERR.ITEMEXISTS `Item already exists`

When calling **AddChild** (or **AddChildren**), the child contact is already a child of the specified parent.

15 ERR.INVPARENT `Parent is not of appropriate type`

Some objects can only be made children of certain types of object. For example, a **ListBox** cannot be made the child of an **ExclusiveGroup**. Refer to Chapter 3 for details of which objects can have which types of parent.

16 ERR.DRAW.ORPHAN `Cannot Draw an orphan`

When calling the **Draw** subroutine, you have specified an object that does not have a parent. Check that you have specified the correct object.

21 ERR.COORDMODE `Invalid coordinate mode`

You have used an invalid value when setting the co-ordinate mode. Refer to the description of the **SetCoordMode** subroutine.

24 ERR.NOSCREEN `Couldn't get the default screen`

You have attempted to obtain the size of the screen by calling **GetSize** and specifying the application context. This has failed, probably because of insufficient memory or Windows resources on the PC. Close down as many applications as possible and try again. If this fails, try restarting Windows.

27 ERR.NOTREALISED `Contact has not yet been realised`

You have attempted to draw text or graphics, or set the cursor position, in a contact that does not have a parent.

35 ERR.FILEOPEN `Error opening file`

The resource file you have specified when calling **LoadAppRes** cannot be found. This might be caused by the following:

- The file name you have specified is incorrect.
- The directory you have specified is incorrect.
- The directory specified in the RFW.INI file does not contain the resource file you have specified.

Refer to the description of the **LoadAppRes** subroutine for details of how to load resources.

38 ERR.INVFILENAME `Invalid file name specified`

The file you have specified when calling the **LoadAppRes** subroutine has the wrong extension. Resource files must have the extension '.RES'.

501 ERR.CLIP.FORMAT `Format is not available`

You have attempted to use a clipboard format that is not supported by this version of UIMS. Refer to the desciptions of the **ClipboardGetContent** and **ClipboardSetContent** subroutines for details of supported formats.

502 ERR.CLIP.OPEN `Failed to open clipboard`

Another Windows application has the clipboard open. Only one application can use the clipboard at a time.

603 ERR.EBOX.NOTEXTSEL `No text selected`

You have attempted to cut or copy selected text from an **EditBox** or **TextEditor** (**Cut** or **Copy** subroutine called with start and end parameters all set to -1), but there is no text selected in the specified contact.

800 ERR.DLGBOX.INVMODE `Invalid dialog box mode`

You have used an invalid value when setting the mode of a dialog box. Refer to the description of the **DlgBoxSetMode** subroutine.

DDE Error Codes

These codes are defined in the item UIMS-DDE in the file UIMS-TOOLS.

Note that the **GetErrorText** subroutine returns the description "Unknown" for all these errors.

Any code other than those listed below indicates an internal error.

0 ERR.RFW.SUCCESS

Subroutine completed successfully.

2003 ADV.CONFAIL

A permanent DDE link was not established or has been terminated by the server.

2010 ERR.DDE.CONFAIL

An attempt to initiate a conversation failed. This might be for any of the following reasons:

- The server application could not be found.
- The server application does not support DDE.
- The specified topic was not recognised by the server.

2101 ERR.DDE.BUSY

The server was unable to respond because it was carrying out another task.

2107 ERR.DDE.LOW.MEMORY

There is insufficient memory available because of an internal error condition.

2108 ERR.DDE.MEMORY.ERR

UIMS was unable to allocate the memory needed for the current task.

2114 ERR.DDE.SERVER.DIED

The server has attempted to continue a conversation that has been terminated by the client., or the server terminated before completing a transaction.

2115 ERR.DDE.SYS.ERR

An internal error has occurred.

Execute Error Codes

These codes are defined in the item RFWDEFS in the file UIMS-TOOLS.

Note that the **GetErrorText** subroutine returns the description "Unknown" for all these errors.

0 ERR.RFW.SUCCESS

Subroutine completed successfully.

8100 ERR.EXECUTE.MEMALLOC

Out of memory.

8102 ERR.EXECUTE.NOFILE

File not found.

8104 ERR.EXECUTE.NOPATH

Path not found.

8105 ERR.EXECUTE.LINK

Attempt to dynamically link to a task.

8106 ERR.EXECUTE.DATASEG

Each task requires a separate data segment.

8110 ERR.EXECUTE.WINVERSION

Incorrect Windows version.

8111 ERR.EXECUTE.INVEXE

Invalid Windows executable file (non-Windows application or error in .EXE image).

8112 ERR.EXECUTE.OS2

OS/2 application.

8113 ERR.EXECUTE.DOS

DOS 4.0 application.

8114 ERR.EXECUTE.INVEXE2

Unknown executable type.

8115 ERR.EXECUTE.OLDEXE

Windows program not supported in current mode.

8116 ERR.EXECUTE.RUNNING

Attempt to run a second instance of a program containing multiple, writeable data segments.

8117 ERR.EXECUTE.RUNNING2

Attempt to run a second instance of a program that links to non-shareable dlls.

8118 ERR.EXECUTE.PROTECTED

Attempt to run a protected mode application in real mode.

8132 ERR.EXECUTE.INUSE

Application already in use (only applies if the *Control* parameter includes the **EXECUTE.SINGLE** option).

8133 ERR.EXECUTE.MEMLOCK

Internal error.

SendKeys Error Codes

These codes are defined in the item RFWDEFS in the file UIMS-TOOLS.

Note that the **GetErrorText** subroutine returns the description "Unknown" for all these errors.

0 ERR.RFW.SUCCESS

Subroutine completed successfully.

8135 ERR.SENDKEYS.FAIL

Internal error.

8136 ERR.SENDKEYS.INUSE

SendKeys is in use by another instance of RealLink.

SystemCommand Error Codes

These codes are defined in the item RFWDEFS in the file UIMS-TOOLS.

Note that the **GetErrorText** subroutine returns the description "Unknown" for all these errors.

0 ERR.SYS.SUCCESS

Subroutine completed successfully.

8140 ERR.SYS.INVCOMMAND

Illegal command.

8141 ERR.SYS.FAIL

Command failed.

8142 ERR.SYS.DIRECTORY

A directory with the specified name exists.

8143 ERR.SYS.NOFILE

The file or directory does not exist.

8144 ERR.SYS.NOTUIMS

You have attempted to call the **SystemCommand** subroutine while using a character-based terminal or a terminal emulator other than RealLink for Windows.

NewView Error Codes

These codes are defined in the item RFWDEFS in the file UIMS-TOOLS.

Note that the **GetErrorText** subroutine returns the description "Unknown" for all these errors.

0 ERR.RFW.SUCCESS

Subroutine completed successfully.

8201 ERR.NV.NOMEM

NewView was unable to allocate the memory required for an operation. This is usually caused by insufficient memory or Windows resources on the PC. Close down as many applications as possible and try again. If this fails, try restarting Windows.

8202 ERR.NV.EXISTS

You have attempted to create a NewView group with an identifier that is already in use by another group. Refer to the descriptions of the **CreateNVContactGroup** and **CreateNVHotspotGroup** subroutines.

8203 ERR.NV.INVALIDID

You have used an invalid NewView identifier – no group exists with the identifier you have specified. The group might not yet have been created, or might have been destroyed.

8204 ERR.NV.INVALIDINST

You have used an incorrect application context handle or have failed to sign on before calling a NewView subroutine that requires the handle of the application context.

Active Window	The window which the user can currently manipulate or work with. This is similar to having the **focus**.
API	Application Programming Interface.
App Window	An App window is the main type of window in a UIMS application. It is free to appear anywhere on the screen and to overlap any other window (compare **child window**). A UIMS application must have at least one App window (the root window).
Attribute	1. A unique characteristic of an **object** that can be modified. 2. A section of a REALITY file item, delimited by attribute marks – CHAR(254).
Brush	1. The way the interior of a graphical object looks; it can be coloured, hatched, or patterned. 2. A UIMS **object** that controls these characteristics.
Check Button	A check button is a **control** that can be turned on or off and saves its state. It looks like a square box to the left of some text. If it has been selected, an 'X' appears in the box.
Check Mark	A mark shown beside a menu item to indicate a selected option. The mark displayed is normally a tick (✓), but on some hardware platforms other marks may be used.
Child Window	A child window is similar to an **App window**, but cannot overlap windows other than its parent.
Client Area	The client area is the part of a window where an application can draw. It is usually the central area of the window and excludes the title area, menu bar, scroll bars, etc.
Client Coordinates	Coordinates relative to the top left-hand corner of the window's **client area**.
Clip Region	Defines in which part of a window drawing can take place. An application may draw outside the clip region, but only the part inside the clip region will be displayed.
Clipboard	The clipboard can be thought of as a resting place in memory for data that has been copied or cut from one application to be pasted into the same or a different application.
Contact	An **object** that provides an interface with the user. Window, menu, and dialogue box objects are all contacts.

Context	An **object** that defines certain application-wide parameters, such as the coordinate mode, the default drawing objects, and the event mask.
Control	A control is a **contact** that carries out a specific kind of input or output. Edit boxes, titled buttons and scroll bars are examples of controls.
Copy	To Copy means to get data from an application and put it in the **clipboard**.
Cursor	A blinking graphic entity that shows where the next text input will appear on the screen.
Cut	To Cut means to get some data from an application and put it in the **clipboard** and then to remove the data from the application.
DDE	Dynamic Data Exchange – a message exchange protocol used in the Microsoft Windows environment.
Default Titled Button	A default **titled button** is a control that represents the usual response to a request. It has text surrounded by an emboldened rectangle. If the user types the RETURN key it is the default titled button that takes effect.
Dialog Box	A dialog box is a window that an application displays to request information from the user. It contains **controls** that the user can manipulate.
Disabled	If an application does not want to allow the user to select a particular option at a certain time, it can disable the option. Disabling a contact causes any text in the contact to be greyed.
Edit Control	A **control** that lets the user type in his own text.
Enabled	Selectable by the user.
Event	Actions carried out by the user result in UIMS events, the details of which are sent to the application by means of **messages**. For example, when the user presses a key, the resulting event generates a keypress message, which tells the application which key was pressed.
Focus	If a window has the focus, all keyboard events will be sent to that window.
Font	The typeface used to display text.
GUI	Graphical User Interface.
Instance	An occurrence of an application.

List Box	A list box is a **control** that presents the user with a list of options which may be clicked on to accomplish some action. Often there is a **scroll bar** attached to the list box to scroll through the options, which may be numerous. A common use of a list box is to present the user with a list of files to select from.
Menu	A menu is a list of action choices listed at the top of a window that can be selected with a pointing device or from the keyboard.
Message	UIMS communicates with applications by passing pre-defined formatted messages. Examples are messages which tell the application to paint its window, and messages which tell the application that the user has selected a command on the menu.
Object	A software packet containing a collection of related data (in the form of attributes) and procedures for operating on that data.
Option Button	An option button is a **control** that usually appears in a group of other option buttons. Each choice is mutually exclusive of the others in the group, so that once the user selects one button, any other button in the group turns off. Selecting a option button is analogous to selecting a radio station on a car radio; for this reason, option buttons are often called radio buttons.
Parent	An **object** or **contact** to which other objects or contacts are attached. For example, a dialog box is the parent of the controls it contains.
Paste	A command to insert the current contents of the **clipboard** into an application's window.
Pen	The way the outline of a graphical **object** looks. It can be wide, coloured, or patterned.
Pointer	A graphic entity that is controlled by a **pointing device** to make selections in an application's window.
Pointing Device	A pointing device is an input device used to control the **pointer** on the screen. It can be a mouse, a light pen, a joystick or a graphic tablet.
Resource Compiler	The resource compiler converts a text file that describes the resources (menus, dialog boxes, etc.) used by an application into the format required by the application.
Screen Coordinates	Coordinates relative to the top left corner of the display.
Scroll Bar	A scroll bar is a **control** that allows the user to set analogue values. Its main use is to let the user change the current view of the application when there is more data than can be displayed in one window.

System Menu	The system menu is a special menu that is pulled down from the top left corner of a window. It contains actions that are usually common to all applications such as moving or changing the size of the window.
Thumb	A part of a **scroll bar** that can be dragged with the mouse to change the scroll bar setting. Its position on the scroll bar indicates the current setting.
Title Bar	The title bar is the uppermost part of a window that provides two pieces of information; the name of the application and whether the window is currently active. Another name for the title bar is the caption bar.
Titled Button	A titled button is a **control** that has text surrounded by a rectangle. Clicking on it causes an immediate reaction. For example, in dialog boxes there are OK and Cancel buttons. Titled buttons are also known as Push Buttons.