OpusSCRIPT User Reference

Digital Workshop

Welcome

The first part of this manual provides an introduction to OpusScript and the various ways you can use scripts in Opus.

The second part of the manual provides a comprehensive script reference taken from the help file. It lists all the script functions in their relevant categories and provides a simple example of the syntax used for each.

OpusScript Help File

Opus Pro provides a separate help file for OpusScript from which this material was taken. If you want to review this material on screen then you can open the OpusScript Help file as follows...

Method 1:

1. Select OpusScript Help option from the Help menu at the top of the Opus Editor – this will open the OpusScript Help file showing the Overview of OpusScript topic.

Method 2:

1. In the Script Object window or the Script Action's Script Editor pane, press the function key F1 – this will open the OpusScript Help file showing the Overview of OpusScript topic.

Method 3:

1. Select an OpusScript function name from the Script tab in the Organiser on the left-hand side of the Opus Editor.

2. Right-click on the function name to open the right-click menu.

3. Select Help from the right-click menu – this will open the

OpusScript Help file at the function's topic page.

Links to OpusScript Help

To provide as much information and help as possible, the main Opus Pro Help file contains links to the OpusScript Help file. These are indicated by a Script icon beside the link name, for example...

This example appears in the Show Action topic in the main Opus Help file and will open the Show() topic page in the OpusScript Help file when it is pressed.

This is particularly useful for two reasons: firstly, it indicates which actions have an equivalent OpusScript function; secondly, you can quickly jump to the relevant page by clicking the link.

The Organiser

On the left-hand side of the Opus Editor is the Organiser. This allows you to view your currently opened publications in different ways by clicking on the tabs on the left-hand side of the Organiser. In the illustration below, the Objects tab has been selected...

A Script tab has been added to the Organiser that contains a handy index of all the OpusScript functions you can add to a Script Object or Script Action in your publication.

The Script tab

The Script tab contains different categories of objects in your publication, listed alphabetically e.g. Basic Objects, Boolean, Browsers, Buttons, etc. Each category contains the OpusScript functions that relate specifically to that category, for example, there are two OpusScript functions related to Button objects, these are listed under the Buttons category, as illustrated below...

The Script tab is useful for four reasons:

(i) Quick Reference – you can immediately see what OpusScript functions are available for different objects. For example, if you want to add a clock object to a publication, the Clock category lists all of the OpusScript functions related to clocks available to you.

(ii) Easy to find – within each category, the OpusScript functions are listed alphabetically, thus making it easy to quickly find an OpusScript function.

(iii) Insert directly into a script – if a Script Object window or a Script action's Script Editor pane is currently open, you can double-click on an OpusScript function name to add it to your script. The new function will be added at the point where the cursor is currently flashing in the script – this is the skeleton of the code, you will have to fill in the details, such as parameters, if required..

(iv) View OpusScript help – if you select an OpusScript function name from the Script tab and right-click on it, the right-click menu will open. This menu contains a Help option that will open the OpusScript Help file at the selected function's topics page. The Help page contains a detailed description of the function, its use and parameters required, as well as a link to examples of the function in use.

Hierarchy of Objects in Opus

In Opus you can create many types of objects on a page, such as, Image, Text, Button and Frame objects, to name a few. In fact, everything in Opus is an object, including the page, a chapter and even the publication itself.

Therefore, Opus is an object-oriented program. For example, you can add chapters to a publication, pages to a chapter, objects to a page and then change the objects properties and make things happen by adding actions to the object.

When using OpusScript, it is useful to know about the hierarchy of objects in Opus because the OpusScript functions are split into separate folders, such as, Graphical Objects and Slideshow. These folders (or classes of objects to be more accurate) allow you to see which functions you can use with a specific type of object, for example, the Play function in the Slideshow category allows you to start a slideshow in your publication, it cannot be used to play a sound or start a timeline.

However, there are other OpusScript functions that you can use with Slideshow objects that are not listed in this category – but what are they? Well, it all depends on the category's position in the hierarchy of objects, as explained below.

Hierarchy:

1. In the most simplest terms, the hierarchy is shown in the Overview topic for each OpusScript Function folder under the heading Hierarchy – see Hierarchy Example below.

2. The Hierarchy heading shows the other OpusScript functions you

can use for this type of object.

3. The class structure heading shows a graphical representation of the class hierarchy – see Hierarchy of objects in Opus below for more information.

4. The Functions heading in the Overview topic lists all of the OpusScript functions for this folder. Click on the name of the OpusScript function to jump to its help page.

Hierarchy Example:

The illustration below is an extract from the Text Objects - Overview topic showing the hierarchy information...

In the Hierarchy heading, the button shows the links to other OpusScript function folders that you can also use. In this illustration, Text objects can use the functions listed in the Text Objects folder and the Graphical Objects and Basic Objects folders as well. The class hierarchy heading shows a graphical representation of the hierarchy.

In this illustration, Basic Objects is the Parent of Graphical Objects, while Graphical Objects is the Parent of Text Objects.

Hierarchy of objects in Opus:

1. Objects belong to a class of objects. This simply means that objects of the same type belong to the same class, for example, all Text objects belong to the same text class and all MultiFrame objects belong to the same multiframe class.

2. Classes are arranged in a hierarchy. This simply means that some classes belong to another class, which is known as the Parent. For example, the Text Objects class belong to the Parent known as Graphical Objects, therefore, the Text Objects class is a Child of the Graphical Objects class.

3. Classes inherit the functions of their Parent. This simply means that the OpusScript functions listed in the parent folder can also be used by their children.

Note:

For example, the Text Objects folder contains the specific OpusScript functions that can ONLY be applied to Text objects, while its parent (the Graphical Objects class) contains OpusScript functions that can be used by Text objects but also by other children of Graphical Objects, such as Button objects and Slideshow objects.

4. Classes inherit all functions higher up the hierarchy. In other words, if a Child has a Parent and it is itself the Child of another Parent, then the child can use the OpusScript functions of its parent and its parent's parent.

Note: For example, Text Objects can use all of the functions in the Graphical Objects folder and the Basic Objects folder because Graphical Objects is a Child of the Basic Objects class.

5. The Basic Objects class is the Parent of all objects in a publication. The Basic Objects folder is the root of all objects in a publication – the OpusScript functions in this folder can be used by all objects that make up your publication.

6. Some classes belong to the script only. Some folders, such as, Clock, Math and Numbers are the top of their hierarchy, that is, they have no parent or children. In other words, the OpusScript functions contained in these folders can be used in a script but don't actually refer to objects in a publication.

Note: For example, the CreateClock function in the Clock folder allows you to create a clock in a script but it doesn't manipulate a Clock object in your publication. This function will store the clock in a variable in the script, you can then display this variable in your publication to the user.

Script Tooltips and Auto correction

All OpusScript functions are case-sensitive, which means they must be entered using the correct capitalisation. Opus Pro now provides a new auto correction feature when entering OpusScript functions within a Script Object or Script Action. Furthermore, a tooltip will appear when an OpusScript function is entered showing the parameters required.

Auto correction

When an OpusScript function is typed in manually, Opus will automatically correct the case of the function if it is entered incorrectly. For example, if you enter the OpusScript function in as show(), this will be corrected to Show().

The word is automatically corrected as soon as you type in the left-hand round-bracket, at which point a Script tooltip will appear – see below for more information.

Auto correction is a particularly useful feature because scripts often do not work because of something as simple as a spelling mistake.

Note: Only OpusScript functions are automatically corrected - any function that you create will not be auto corrected if used elsewhere within your publication.

Script Tooltips

When an OpusScript function is typed in manually or added using the Script tab in the Organiser, a Script tooltip will appear showing the correct syntax for the function along with a brief description of its function and the Return Value it provides (if applicable).

Below is an illustration of the CreateClock function's tooltip...

When an OpusScript function is added or typed into a script, the parameter names are included within the round-brackets section of the tooltip. In the illustration above, the name parameter is highlighted because it is the first parameter required, the last line of the tooltip provides a brief description of the parameter information required.
Parameters within a function are separated by commas. As soon as you enter a comma in the round-brackets of a function, the next parameter name will be highlighted along with its description within the tooltip.

The tooltip will hide once you type the right-hand bracket to close the function's parameters. Alternatively, if the OpusScript function was entered using the Script tab in the Organiser, click outside the round- brackets of the function and press the Enter key to hide the tooltip.

The Script tooltip feature can be modified or turned-off using the Options dialog from the Tools menu at the top of the Opus Editor – see next section for more information.

Note: Only OpusScript functions show a Script tooltip - any function that you create will not contain a Script tooltip.

Script Options

The Options dialog box in the Tools menu allows you to customize many of the things about the Opus environment, including your own preferences for the OpusScript programming language.

To open the Script page in the Options dialog:

1. Click on the Tools menu at the top of the Opus Editor.

2. Select the Options... option – this will open the Options dialog box containing an Options Organiser on the left-hand side of the dialog that provides an Explorer type view of folders. The options for each page in a folder is displayed in the right-hand side of the dialog.

3. Click on the Script page icon in the Options Organiser – this will display the Script options in the right-hand side of the Options dialog.

There are three areas to the Script page, these are:

1. Script Console panel...

The Script Console is a small dialog that contains any error messages or information you have sent to it using the Debug.trace function. This panel deals with the Script Console and when it appears and what it shows when you preview your publication. The Script Console never displays in the published version of your publication only in the Opus Editor when your preview your publication.

2. Script Editor panel...

The Script Editor is the screen in which you type your OpusScript program. If you use OpusScript functions, you can have a tooltip appear showing the syntax for the selected function – you can edit which settings appear in the tooltip.

3. Colours panel...

The scripts you type in the Script Editor are colour coded for different types of entries, such as Keywords and Strings. You can edit the colour settings for the different types of entries to colours you prefer – this may be the same colours as you have set in other programming tools, such as an HTML Editor.

To edit the Script Console panel settings:

1. Use the Show console when preview starts option if you want the Script Console to automatically show when you preview the page.

2. Use the Show console when any output is written option will show the Script Console on screen when a Debug.trace function is written in your script.

3. Use the Show console when “Errors” pane is written to option will show the Error tab of the Script Console on screen when your script does not work as expected, e.g. a variable cannot be found.

4. Use the Hide the console when the preview finishes option if you want the Script Console to close when you return to the Opus Editor from previewing your publication. As the Script Console contains error messages that you will need to follow up on when in the Opus Editor, we recommend you don’t use this option.

5. Use the Clear the console when preview starts option if you want any previous messages displayed in the Script Console to be removed before you preview your publication. It is useful to start with a clear console so that you know any messages in the console refer to your current preview.

6. Use the Switch to output pane option if you want the Output tab

to appear as the front tab in the Script Console.

To edit the Script Editor panel settings:

1. Use the Show function tips option if you want a small tooltip like this to appear...

Note: If this option is not ticked, the tooltip illustrated above will not appear and the rest of the options in this dialog will not work either.

2. Use the Show return information option if you want the tooltipto include the information your function will return.

3. Use the Show extended description option if you want the text

description displayed in the tooltip along with the syntax.

4. Use the Show parameter information option if you want the

parameters inside the brackets included in the tooltip.

To edit the Colours panel settings:

1. Select the name of the script entry type (e.g. Keyword) from the

list if you want to change its colour.

2. Use the drop down colour well to select the new colour for the script entry type you selected – the Example Text box will show the new colour selected.

Script Templates

The Script Templates option in the QuickBuild menu allows you to insert pre-written OpusScript programs into your publication by simply selecting the name of the script template from the list provided. This means you can re-use your code over multiple projects without having to re-write or import your code from another publication.

Note: At present the Script Templates option contains one script template. However, new scripts will be added in future versions of Opus. Don't forget, you can add your own scripts at any time to the list – see below for more information.

To add a Script Template to an object or a page:

1. Select the page or the object on the page that will use the script template.

2. Select Script Templates from the QuickBuild menu – the Script Templates dialog will appear on screen.

3. From the Script Templates dialog, select the name of the template you want to add.

4. Tick the Use a copy of the script option if you want to insert an unchangeable version of the script. By default, this option is ticked. A warning dialog box will appear if you untick this option.

Note: If this option is not ticked, you can edit the inserted script template within your publication. Any changes you make will overwrite the original script template. Make sure you only untick this option if you want to edit the original file, which will in turn affect all uses of that script.

5. Click the OK button to add the script template. This will insert a Script object containing the script to the object you selected in point 1 above.

To add new scripts to the Script Templates list:

The Script Templates tool is very useful if you regularly use OpusScript to perform tasks in your publication. Adding a script template to the list does require you to know in which folder Opus was originally installed on your machine. If Opus is available via a network, you may need to contact your Network Administrator for more information or permission to add your script template.

1. Insert a Script object on a page and type in your script. This can be as short or long as you require. Try and make the script generic so that it can be used by as many of your publications as possible.

2. Click on the Set External File icon at the top of the Script Object window – this will open the standard Windows Open dialog.

3. In the Windows Open dialog's File name box type the name you want to give to your script. Make the name descriptive as it is the filename that appears in the list of the Script Templates dialog.

4. Locate the folder where Opus was originally installed and open the sub folder named Script Templates. For example, on a Windows 2000 machine, Opus is automatically installed in C:\Program Files\Opus Pro 04 folder.

Note: A copy of the saved script file MUST be stored in the Opus sub folder Script Templates as this is the only place that Opus generates the list that appears in the Script Templates dialog.

5. Click the OK button to save the external file. The new script file

will be given the extension name ILS.

6. Before your script will appear in the Script Templates dialog's

list, you must exit from Opus and then restart it.

7. Finally, follow the To add a Script Template to an object or a page instructions above to add your script template to any of your publications.

Note: For more information about creating an external file for a Script object, select OpusScript Help from the Help menu at the top of the Opus Editor and search for the phrase External Scripts.

Layout of functions

The description of each OpusScript function is split into sections. The sections are:

Syntax:

The Syntax section shows the function name and the case in which it should be typed e.g. GetLineCount(). You must type the name of the function without spaces between the words and with the round brackets.

Some round brackets contain parameters e.g. SetColour(Colour), in which case a parameter section will also appear on the help page for the function – see Parameters below for more information.

Note: When you type a script in the Script Action or Script Object, Opus will automatically correct the case of the OpusScript functions for you. Also, a tooltip will appear beside the name of the function showing a brief description of the function and the parameters required.

Return:

This section will only appear on a help page if the function returns information to the user e.g. IsAutonarratePlaying().

The Return Value will describe the type of information that is sent back when this function is executed. The return may be a number, a Boolean value, a string or a new object. Make sure you understand if the function is returning information and what type of information it returns. Often you will be using the return information to perform other functions within your script.

Example:

```
var textPlaying = IntroText.IsAutonarratePlaying() If (textPlaying == true) {
  ImageIntro.Show() }
```

As in the example above, if you get a Return Value, you must use the function inside a variable. In the example above, the variable

textPlaying contains the return value for the IsAutonarratePlaying function. The if function then uses the value in textPlaying to show an image only if the Autonarration is playing.

If a function starts with the word Is (e.g. IsAutonarratePlaying, IsObjectIntersecting and IsEnabled) then you will get a return value – this will most likely be a Boolean value i.e. true or false.

All of the functions beginning with Get (e.g. GetWidth and GetPosition) will return a value. So make sure you assign these functions to a variable, like the example above.

Some of the functions that begin with Get (e.g. GetAppearance and GetSelectionStyle) will return a new object. These objects have their own properties that you can use.

Nearly all of the Math and Date functions will return a value.

Some other functions will return a value as well – these are always indicated on the help page for the function.

Parameters:

Each parameter in a function is given a description in the Parameters section. This indicates what type of information is required for the parameter, such as a string, a pathname or a number.

If a function contains more than one parameter, each parameter is separated by a comma. You must enter the parameters in the order they appear in the Syntax section. For example, SetSelection has two parameters, start and end you must enter the value for start first followed by a comma and then by the second value.

Each parameter description tells you if the parameter is required or optional. If it is required, you must include a value in the function. If it is optional you do not have to give a value.

Some parameters have a default value, this means that a value will be used if you do not enter your own value. If a parameter is required but has a default value, you do not have to enter the value. However, if you have several parameters in a function each with a default and you want to change only one of the values (for example the third parameter) you must include any preceding parameters; even if you simply use the default values.

Example:

If a hypothetical function:

```
ExampleFunction( A, B, C )
```

takes A (a number, default 10), B (a boolean, default false) and C (a string, default "bob"); then to call the function with these defaults simply write:

```
ExampleFunction()
```

If you want to override the value for C then you must specify A and B first, even if you use the default values:

```
ExampleFunction( 10, false, "fred" )
```

Simply entering

```
ExampleFunction( "fred" )
```

will not work. The string "fred" will be interpreted as parameter A and given the valueOf of "fred" – the remaining parameters B and C will be given their default values.

Remarks:

This is a brief description of the task the function will perform. This may also include other information, such as related functions or if another function must be used before you can use your function.

Syntax for Data Types

OpusScript supports six types of data types: numbers; boolean values; strings; objects; null; and undefined. A variable can contain any of these data types.

Numbers:

Numbers can be either Integers or fractional numbers (also known as floating-point numbers). In practical terms, you do not have to worry how you enter a number unless you are going to do more complex mathematical or scientific calculations.

Example 1: Integers

```
1 39268
```

Example 2: Floating-point numbers

```
3.1415 .001
```

Boolean values:

There are only two types of Boolean values: true or false. These values equate to yes and no and are the backbone of any decision-making statements you enter in an OpusScript. The Boolean values true and false are colour-coded blue in an OpusScript for ease of identification.

In the example below, the variable passedText is set to the Boolean value true. The expression in the if statement is therefore true and the passImage.Show() statement will run and not the failImage.Show() statement

```
var passedTest = true if (passedTest) {
      passImage.Show() } else {
      failImage.Show() }
```

String:

A string is any combination of alphabetic or numeric characters surrounded by single or double quote marks. The string must be surrounded by the same style of quote marks i.e. single or double. One style of quote marks can be contained within the other style. A string is colour-coded gold in an OpusScript for ease of identification.

Example 1: Simple strings

```
var name = "John Smith" var code = 'OX16 OTH' var poem = "Tam O' Shanter"
```

Example 2: Combining Strings One string can be appended to the end of another string using the + sign

```
var firstName = "John" var surname = "Smith" var fullName = firstName +
surname
```

When a variable containing a string is added to a variable containing a number, the resulting variable is a string data type, as follows

```
var first = "Room" var second = 101 var result = first + second // result =
"Room 101"
```

Example 3: Numbers in strings

When numbers are surrounded by quote marks, they are considered a string

```
var x = "12" + "34" // x = "1234"
```

Example 4: Special characters Some important special characters can be inserted into a string by using the backslash character (\ ). These are:

(i) \n – a new line. This is useful if you are using the Debug.trace function to check your script because you can separate information from variables on different lines.

```
Debug.trace( "The answer are: " + answerA + "\n" + answerB)
```

(ii) \t – a tab. This is useful if you want to insert a tab in a file opened

with the OpenFile function.

```
var textObj = OpenFile("c:\\myText.txt") writeString = "John" + "\t" +
"Smith" + "\n" textObj.WriteLine(writeString)
```

(iii) \\ - a backslash character. If you need to write a pathname, you will need to enter double slashes to indicate sub-folders in the path.

```
var graphicPath = "c:\\Images\\businessGraphic.png"
```

Object:

A variable can also hold the values of a new object. This object can have properties that have been added to it when you have used a Get function, such as, GetSelectionParagraphStyle. Alternatively, you can create your own object and give it properties – this is commonly done if you want to set new properties using some of the Set functions, such as, SetSelectionParagraphStyle.

Example 1: Store properties of an object

In this example, we will store the Return Values of the GetSelectionStyle function into a new variable named textStyle

```
Text_1.SetSelection(0,-1) var textStyle = Text_1.GetSelectionStyle()
```

The first line of code selects all of the text in a Text object named Text_1. The second line shows a variable named textStyle which holds the object returned by the function GetSelectionStyle.

Example 2: Store property into a variable The new object named textStyle (created in Example 1 above) has a number of properties. By adding this next line of code you can store one of the properties of an object into another variable. For example, to store the value of the property fontname (a property of the returned object for the GetSelectionStyle function), use the following syntax:

```
var OldStyle = textStyle.fontname Debug.trace(OldStyle)
```

The OldStyle variable will contain a string showing the name of the font used for the Text object Text_1 e.g. "Arial". The second line of code shows the value of OldStyle in a Debug window. In practice, you won't want to show the variable in a Debug window, you are more likely to want to use other functions to interrogate the data. For example, you may want to check if the font used is Arial – if it isn't Arial you can change the font to Arial (see Example 3 and 4 below).

Example 3: Create your own object

Often you may want to create your own object – this is normally when you want to change the data returned in an object when you have used a Get function. First lets look at how you create a new object and add properties to the object

```
var SpaceShip = new Object() SpaceShip.Height = 50 SpaceShip.Width = 30
```

The first line of code creates a new object. The syntax is always the same whenever you create a new object, that is

```
VariableName = new Object()
```

The VariableName will change for each new object (in Example 3 the variable name is SpaceShip) but everything after the equal sign remains the same. The second and third lines in Example 3 create two new properties for the object, Height and Width which are given a value.

Example 4: Using a new object with other functions Often, you will only want to create your own object if you want to change the data in an object on a page while the publication is running. For example:

```
myText.SetSelection(0,-1) var newStyle = new Object() newStyle.bold = true
newStyle.fontname = "Comic Sans MS" myText.SetSelectionStyle(newStyle)
```

The first line of code selects all of the text in a Text object named myText. The middle lines of code create a new object named newStyle and set two properties bold and fontname. In the last line of code, the properties in the object newStyle are applied to the Text object myText.

Null:

The null type has just one value – null. It is used for advanced scripting, usually to indicate the absence of any particular value.

Undefined:

The undefined type has just one value – undefined. Any variable or property that has not been initialised has the undefined type.

Syntax for Operators

In a script symbols such as + and – perform a function. The plus sign can add two numbers together and the minus sign can subtract one number from another. These symbols are called Operators, and OpusScript provides a wide range of operators, which are listed below.

Operator Precedence

Some operators have a higher precedence than others, which means they will be calculated before other operators. Take,for example, Arithmetic Operators:

```
x = 2 + 3 * 4 / 2
```

The * and / symbols are used for multiplication and division. The * and / operators are of equal precedence, the + and - operators are also of equal precedence. However, the * and / operators have a higher precedence than + and -, which means the * and / operators are calculated before the + and - operators in an expression.

In the example above, the system will calculate the expression as:

```
3 * 4 = 12 12 / 2 = 6 2 + 6 = 8 x = 8
```

If you are in doubt as to what order an expression will be calculated or wish to force a particular order to the calculation use parenthesis. Parenthesis (i.e. round brackets) are calculated before other operators in an expression. For example, if you want x to equal 10 you could do the following calculation:

```
x = (3 + 3) * 4 / 2
```

Note that you can "nest" sets of parentheses, that is you can put sets of brackets inside other brackets.

Debugging Note:

The order operators are calculated can mean forgetting a bracket can change a result significantly, which can cause bugs in your program.

Arithmetic Operators

Basic arithmetic and some more advanced syntax.

Symbol Meaning Example

+ Addition x = 5 + 4 - Subtraction x = 5 – 4 * Multiplication x = 5 * 5 / Division x = 27 / 3 ++ Increment (add one to the value) x=1;

```
while (x < 10) {
   x++; }
```

-- Decrement (subtract one from

the value)

```
x = 10; while (x > 0) {
```

```
x--; }
```

Assignment Operators

An assignment operator assigns a value to the left hand side of the equal sign based on the value in the right hand side of the equal sign.

It is very important to understand the difference between this operator and the comparison operator (= =)

Symbol Meaning Example

`= Assign Y = 10;`

`x = y;`

will result in

`x == 10`

+= Increment by X += y is the same as X = x + y

-= Subtract from X -= y is the same as X = x – y

*= Multiply by X *= y is the same as X = x * y

/= Divide by X /= y is the same as X = x / y

Comparison Operators

A comparison operator compares the left hand side of the equal sign with the right hand side of the equal sign and returns a logical value of true or false based on whether the comparison matches or does not match. You can compare numerical or string values.

Symbol Meaning Example

`== Equals if (2 == 2)`

`=== Strict Equals if (2 == 2)`

`!= Not Equal if (3 != 2)`

`< Less than if (2 < 3 )`

`> Greater than if (3 > 2)`

`<= Less than or equal to if (2 <= 2)`

`>= Greater than or equal to if (2 >= 2)`

Logical Operators

Logical Operators are normally used with Boolean values that return a value of true or false. The Logical Operators return the value of one of the specified expressions.

Symbol Meaning Example

`&& Logical AND x = true && (2 == 2) will set x as true`

```
|| Logical OR x = true || (2 == 2) will set x as true
! Logical NOT x = !true will set x as false
```

String Operators

As well as the comparison operators that can be used on strings, the concatenate operator (+) concatenates two string values together.

Symbol Meaning Example

+ Concatenates strings

```
x = "my " + "string"
```

will result in

```
x == "my string"
```

and

```
x = "my " + "second " + "string"
```

will result in

```
x == "my second string"
```

+= Append a string x = "my ";

```
x += "string";
```

will result in

```
x == "my string"
```

Syntax for Variables

Variables are containers of information and should be declared (i.e. defined) before you use them. Variables can also be initialised when they are declared. To declare and/or initialise a variable use the var statement.

Syntax:

```
var variableName = value
```

Remarks: variableName - is the name you want to give to the variable. The name must begin with a letter but can contain numeric characters. A name cannot contain spaces, use the _ character if you want to space out a variable name e.g. my_first_name.

value – is the contents of the variable.

Advanced:

When var is used inside a function, it makes the variable local to the function. This means that the variable cannot be used outside the function. If a variable is used without a var declaration, the variable is a property of the page and can be used outside the function or in a different function. Unless you specifically want this to happen, it is probably a bad thing to do, and hence var should be used.

Outside a function var has no effect at all. All variables outside of functions are properties of the page.

Example 1: Declare Variables

```
var myVariable var but_1
```

Example 2: Initialised variables:

```
var w = "The Total value is: " // a string value var x = 22 + 43 //x = 65 var
y = x + 35 // y = 100 var z = w + y // z = "The Total value is 100"
```

Syntax for Maths Functions

The Math object allows you to perform a variety of scientific calculations in your script including functions for logarithms, trigonometry and geometry.

The Math functions provided in OpusScript are all part of the Math object and are either (i) Math constants, such as PI (which maintains a constant number i.e. approximately 3.1415); (ii) or Math methods, such as acos (which you have to provide with information to calculate a result).

Syntax:

var varName = Math.functionName

Remarks:

var - is used to create a new variable in which the Math object will be stored.

varName – the name you want to give to the variable.

Math – the Math object – this must be followed by a full stop and then the name of the math function you want to perform.

functionName – the Math function name you want to perform – see Math Functions Overview for further details.

Example 1: Math constants

The Math object provides a number of constants often-used in trigonometry and geometry, such as PI (the ratio of a circle to its diameter) and logarithms such as E (Euler's constant, used for natural logarithms). All Math constant functions are written in uppercase e.g. PI and SQRT2. When using Math constants you do not have to provide any parameters as the value is always the same. For example, to calculate the area of a circle, use the following syntax:

```
var radius = 32

var area = Math.PI * radius * radius
```

In the example above, the Math constant function PI is multiplied by the radius squared and the result stored in a variable named area.

Example 2: Math methods As well as Math constants the Math object provides Math methods, these allow you to perform a variety of scientific calculations, such as calculating angles with cos (cosine), sin (sine), tan (tangent) and acos, asin, atan and atan2 methods. Other Math methods allow you to perform scientific calculations such as sqrt (find the square root of a number) and pow (raise the power of one number by another). The rest of the Math functions have a variety of uses, for example, floor will round a number to the lowest integer (e.g. 3.456 will be returned as 3), while ceil does the opposite and raises a number to the highest integer (e.g. 3.456 will be returned as 4).

In the example below, the round function is used to round a number to its nearest integer

```
var myNum_1 = 12.49999 var round_1 = Math.round(myNum_1) // returns 12 var
myNum_2 = 12.51111 var round_2 = Math.round(myNum_2) // returns 13
```

Note: A simple rule to remember is that with Math constants, the functions are in uppercase letters and they do not require a parameter. With Math methods, the functions are in lowercase

and require parameters.

Syntax for Opus Objects

Objects are the basis of the OpusScript programming language. Anything that you can add to a page in an Opus publication, such as, a button, image, sound or timeline is an object. Each of these objects can be referenced in OpusScript via their name, as it appears in the Organiser.

Syntax:

```
objectName.function
```

Remarks:

objectName – The name of the object as it appears in the Organiser. The objectName must NOT be surrounded by quotes.

Full stop – The full stop (.) is very important because it separates the object name from the function name.

function - The name of the function you want to apply to the object. This will either be your own function or one of the OpusScript functions.

Examples:

Below are two of how to identify the object name and the function name in a script. The Help page for each function provides at least one example of how to correctly enter the object name and function

```
myImage.Show() button_1.Hide()
```

Note: If the name of the object in the Organiser contains spaces, or any other character not allowed in script variable names e.g. button 1, the spaces must be replaced by the _ character e.g. button_1 or the object will not be recognised by OpusScript.

Syntax for Arrays

An array is like a series of variables. It can hold different individual pieces of information and you can retrieve the pieces of information individually. These segments of information are called the elements of the array. You access each element by its number in the array.

Note: In keeping with standard programming practice arrays start counting from zero. Loop counters also start at zero. If you want to start counting your array elements at 1 you will need to make the array 1 larger than it should be and ensure that loops you use with the array also start at 1.

Syntax:

To create (i.e. declare) an Array

```
var arrayName = new Array()
```

or

var arrayName = new Array(number)

Remarks:

var – the keyword var is used to declare and initialise a new variable.

arrayName – the name you will give to the Array.

new Array() – the keyword new is used to create a new object and Array is the object type you are creating.

() – the brackets contain the number of elements in the array. If the number of elements is omitted, the number of elements can be expanded as required.

Syntax: To use (i.e. reference) elements of an Array: arrayName[elementNumber]

Remarks:

arrayName – the name you gave to the Array.

elementNumber – the number of the element within the array. The first element is number 0, the second is 1 and so on.

[ ] – the elementNumber must be surrounded by square brackets.

Note: You can use the value of an element in a statement.

Arrays

Creating Arrays

When you create an Array, you are said to be declaring the array. In this example a new array is created using the variable day and it will contain seven elements

```
var day = new Array(7)
```

You are not required to put a number in the Array keyword. There are many times when you won't know how many elements there will be in an array, so you can always just create an empty array and populate it, as and when you need to. For example:

```
var day = new Array()
```

Example: Populate the array

When you first declare an array you can populate each element or you can populate an array at any time. The array name is suffixed with a square bracket containing the element number. In practice with standard programming, arrays start with 0 as the first element. For example:

```
day[0] = "Sunday" day[1] = "Monday" day[2] = "Tuesday" day[3] = "Wednesday"
day[4] = "Thursday" day[5] = "Friday" day[6] = "Saturday"
```

Example: Populate an array with a loop

A common method of populating an array is by using a loop. For example:

```
dollars = new Array() sterling = new Array() sterling[1] = 23 sterling[2] =
17.50

sterling[3] = 75 var exchangeRate = 1.70 var loopCounter = 1 while
(loopCounter <= 3) {

      dollars[loopCounter] = sterling[loopCounter] * exchangeRate

      loopCounter++ }
```

In the example above, the array named sterling is multiplied by the value of exchangeRate and the calculated result stored in an empty array named dollars. Notice that the element number can be a variable (as long as the variable is a number). The first time the while loop is executed the variable loopCounter is 1, therefore the element sterling[1] is multiplied by exchangeRate (i.e. 23 multiplied by 1.70) and the calculated result stored in the element dollars[1].

Example: Using an element in a statement

Each element in an array is like a variable, it can contain any of the Data Types, such as a string, a number or a Boolean value. As a result elements are often used in other lines of codes in your program

```
var today = new Array() today[0] = "Sunday" var firstDay = day[0] var
checkDay = "Sunday" if (checkDay == day[0] {

      sundayImage.Show() }
```

The third line in Example 4 above stores the value of element day[0] to a new variable named firstDay (i.e. firstDay now contains the value "Sunday"). In line 4, the variable checkDay is set to the string "Sunday" and the if expression checks if the variable checkDay and the element

day[0] contain the same data, if they do the Image object named sundayImage is displayed on the page.

Functions

A function is a list of codes that can be defined once and then used repeatedly. In OpusScript, it is good practice to write functions in the Script Object and refer to it from a Script Action.

Syntax: To create (i.e. define) a function function functionName(parameters) {

list of statements }

Remarks:

function – is a keyword, it declares the function.

functionName – the name you will give the function.

parameters – the additional parameters you can pass the function from a calling function.

list of statements – the lines of code that will run when the function is called.

{ } – the lines of code must be surrounded by curly brackets, to indicate the start and end of the statements within the function.

Syntax: To call a function

functionName(parameters)

Remarks:

functionName – the name of the function.

parameters – a list of parameters, separated by commas that are passed to the function for processing.

( ) – the parameters must be surrounded by round brackets.

Create a function

To create a function (or in programming terms 'Define' a function) is straight forward, as long as you remember the correct syntax, such as remembering to use the word function provide a function name and parameters (optional) and insert both curly brackets { } one at the beginning and end of the function, you should have no problem. Functions are very useful because they can be 'called' by any object on a page (see Example 2 for more information) as many times as required. For example:

```
function CurrentScore(imageName) {
imageName.Show() var userScore = UserName + " your current score is "
      + SCORE_TOTAL feedback.SetSelection(0,-1)
feedback.ReplaceSelection(userScore) }
```

In this example, the function is named CurrentScore and its function is to tell the user their current score. Imagine your publication has a quiz page with several multiple choice questions on it. Each time the user presses an answer button, a Script Action on the button is triggered and it calls this function (see Example 2 below for more about calling functions). By placing the function in a Script Object, each button can call this function.

Example 2: Calling a function

To call a function (i.e. to run the list of instructions in a function) is really simple, all you need to do is to type the name of the function either in a Script Object or in a Script Action. To call the function in Example 1, use the following syntax:

```
CurrentScore(ImageCorrect, "John")
```

Because the function CurrentScore was defined as taking two parameters, you need provide those two pieces of data when calling the function. In this example, the first parameter in the function was named imageName and it is expecting an Image object on the current page; so when we call the function we enter the name of the image we want to send to the function, in this example an Image object named ImageCorrect. The first line of the function uses this information to show the image on screen. The second parameter in the function is UserName and we send the name of the user; in our example this is a string containing the name John – we could have entered a variable in here that contained the name of the user if we wanted to. The string is then used in the second line of the function.

Using the two above, we have briefly demonstrated how you can create a function and call a function. By passing parameters to a function you can make it carry out its list of instructions using different data each time. This is a very important part of programming; you want to make your code as generic as possible so that it can be used as often as possible.

Note: If you never call a function, then the list of instructions in the function will never run.

Syntax: if and else

The If statement will execute a list of statements when the expression is true. An else statement can be added to an if statement and will execute an alternative list of statements when the if statement is false. An if statement can be nested inside another if statement. The if statement allows a program to make decisions about which set of statements it should run.

Syntax:

if (expression) {

list of statements }

or

if (expression) {

list of statements } else {

list of statements }

or

if (expression) {

list of statements if (expression) {

list of statements } } else {

list of statements }

Remarks:'

if – a keyword, used to start an if statement. The list of statements is only executed when the expression is true. An if statement can be nested inside another if statement – when you do this, it is good practice to indent your code to make it more readable.

else – a keyword, used to start a list of statements when the if expression is false.

list of statements – any valid OpusScript statements.

( ) – expressions must be surrounded by round brackets.

{ } – a list of statements must be surrounded by curly brackets.

Example 1: Simple if:

```
var UserName = "John Smith" if (UserName == "John Smith") {
      welcomeImage.Show() }
```

Example 2: If and an else

```
var totalScore = 75 if (totalScore >= 70) {
      passImage.Show() } else {
      failImage.Show() }
```

Example 3: Nested if

```
var totalScore = 75 if (totalScore >= 70) {
if (totalScore >= 90) {
```

```
      honoursPassImage.Show() } else {
      passImage.Show() }
} else {
      failImage.Show() }
```

Syntax: for loop

The for statement is a simple loop that will loop through a series of statements while the test expression is true, at which point the next line after the loop is executed.

Syntax:

```
for (initialise; test; increment) {
  list of statements }
```

Note: When the loop begins the expression 'initialise' is executed. Then the following 3 steps are performed repeatedly:

(i) The expression test is evaluated and if it is false the loop is

complete and the next statement after the loop is executed.

(ii) The list of statements inside the loop is executed.

(iii) The expression increment is executed and then we begin

again at step 1.

Remarks:

for – is a keyword, the system recognises this is the start of a loop.

initialise – is executed before the first iteration of the loop.

test – an expression that is evaluated at the beginning of each iteration of the loop. When it evaluates to false the loop is complete.

increment – a statement that is run at the end of each iteration of the loop.

list of statements – any valid OpusScript statements.

; - the semi-colon must be placed in-between the expressions or the loop will not work.

( ) – the expressions must be surrounded by round brackets.

{ } – the list of statements must be surrounded by curly brackets.

Any of the parameters initialise, test and/or increment may be omitted, as long as the semi-colons are still present. An omitted test is assumed to evaluate to true, thus the loop will go on forever, unless there is a break or return statement inside it.

for

A practical example of using a for loop is shown below. The function calcAvgScore calculates the mean average score for all of the scores stored in the Array named score. The way in which the for loop has been written means that you can keep adding new scores to the array and you will still get the correct mean average. For example, if you added 20 new elements to the array, the loop will know there are 24 elements in the array (score.length calculates the number of elements in an array) and the for loop will repeat the list of statements 24 times.

```
Function calcAvgScore() { var score = new Array() score[0] = 8 score[1] = 8
score[2] = 9 score[3] = 9 averageScore = 0 scoreTotal = 0

for (loop = 0;loop<score.length;loop++) {
```

```
   scoreTotal += score[loop] } averageScore = scoreTotal / score.length
Debug.trace(averageScore)] }
```

Note: The for loop expression works as follows:

(i) loop = 0 is the initialise section of the for loop. This is executed

before the first time (iteration) through the loop.

(ii) loop <score.length is the test section of the for loop. This is evaluated at the beginning of the loop, if the expression is true, then the list of statements in the loop are run. The first time this is evaluated it reads like this. loop is equal to 0 – which is less than score.length because score.length is 4 (i.e. the number of

elements in the Array score) – therefore the answer is true. The loop statements are run because the answer is true.

(iii) Loop++ is the increment section of the for loop. This is done after all of the statements in the loop have been executed, this adds 1 to the variable loop.

The steps above are repeated until loop is equal to 4, at which point the test becomes false and the loop is complete and the statements are not run.

Syntax: while loop

The while statement is another loop like the for loop. However, for is normally used when you know the number of times that you want to run the loop. The while loop is used when you don't know how many times to run the loop.

Syntax:

while (expression) {

list of statements }

Note: When the while loop is executed, the following sequence of operations occur repeatedly:

(i) The expression is evaluated and if it is false the loop is complete and the next statement after the loop is executed. If the expression evaluates to true the body of the loop is executed.

(ii) The list of statements is executed, and then the loop begins

again.

Remarks:

while – is a keyword, used to start a while loop.

expression – is any valid OpusScript expression.

list of statements – a list of OpusScript statements. One of the statements must allow the loop to complete, otherwise you will have an infinite loop.

( ) – the expression must be surrounded by round brackets.

{ } – the list of statements must be surrounded by curly brackets.

while

Below is a practical example of using a while loop. The loop will convert 3 sterling amounts into dollar amounts using the exchange rate of 1 pound sterling to 1.70 dollars.

```
function SterlingToDollars() {

dollars = new Array() sterling = new Array() sterling[1] = 23 sterling[2] =
17.50 sterling[3] = 75 var exchangeRate = 1.70 var loopCounter = 1 while
(loopCounter <= 3) {

            dollars[loopCounter] = sterling[loopCounter] * exchangeRate

            Debug.trace( sterling[loopCounter] + " = " + dollars[loopCounter]
+ "\n" )

      loopCounter++ } }
```

In this example, the while loop starts at the count of 1, which is less than 3, so it runs the list of statements. The first statement does the conversion for the first amount of sterling (the array sterling[1]) into dollars and saves the amount in the dollars array (dollars[1]). The Debug.trace function shows the calculation in a Debug window. The last line of code increments the loop counter by one – this is important! If you do not increment the counter than the loop counter will always be 1 and the loop will run infinitely.

The second time the loop is run loopCounter is set to 2, which is still less than three, so the loop is run again and the second sterling value is calculated and the other lines of code are also run. The third time the loop is run loopCounter is now 3, which is equal to the 3 in the while expression, so this time through the loop is the last time the loop is run. Be careful - when the value in the while statement is matched, the statements in the list are still run for that time. If you do not write the code correctly, you may go through a loop one more time than you want.

Once a while loop has finished, the next line of code is executed.

Syntax: eval

The eval function will evaluate a string or expression and return the result of the code. This function can be used without referencing an object.

eval determines if the argument contains any OpusScript functions. If there are OpusScript functions present, they will be executed and eval will return the value of the last statement (if there is a value). If the argument is a simple expression, it will be evaluated and the value returned.

Syntax:

eval(string)

Remarks:

eval – is the name of the function.

string - is any string that represents an OpusScript expression, statement or sequence of statements. The expression can contain variables and properties of existing objects. This parameter is required.

eval

The eval function allows you to enter a string, expression or variable name that will return the result of the code, in other words it executes the code if it is valid. In the example below, imagine you have created two columns of buttons on a page: the buttons in one column set the value of the variable objName to a different object name on the page (e.g. IntroImage, SecondImage, ThirdImage etc.); the second column sets the value of the variable NextTask to an OpusScript function (e.g. Show(), Hide(), Move() etc.).

The first two lines of the code below show the possible values for the variables objName and NextTask. Next imagine you click a different button on the page that sets the value of a variable named perform – the third line shows the value set in the perform function (i.e. the value will be the following string "IntroImage.Show()"). Finally the last line uses

the eval function to execute the string as if it was a line of code – in other words, it will show the object named IntroImage on the page.

```
var ObjName = "IntroImage" var NextTask = "Show()" var perform = ObjName +
"." + NextTask eval(perform)
```

This example shows how you can use the eval function to execute a line of code that has been built up using strings. It is not possible "execute" a string or variable on its own because it is simply a string of text – this is only possible by using eval.

Syntax: wait

The wait function is used to stop a Script from executing the next line of code until the number of seconds set in the function has completed.

Syntax

wait(Time)

Remarks:

Time - is the time in seconds to wait. Time must be a number or a variable that evaluates to a number. This parameter is required.

Note: This function is useful if you want to sequence a number of events to happen one after the other. This function can be set to zero (i.e. wait(0)), which is often used to give the system time to update the screen.

```
Example 1: Sequencing events Image_1.Show() wait (1) Image_1.Hide() wait (1)
Image_2.Show() wait (0.5) Image_2.Hide
```

Example 2: Refreshing the screen

The wait function can be set to zero, which is often used to update the screen. For example, if you do this

```
for(;;) {
      Image_1.Show() Image_1.Hide() }
```

You may expect the image to flicker, however, the screen is not updated between the Show and Hide. To update the screen, use wait(0):

```
for(;;) {
Image_1.Show() Wait(0) Image_1.Hide() }
```

Syntax: Date

The Date function will return the date and time in a new object, or it can be used to set the date and time in a new object. You can retrieve individual properties of the date or time, such as, day, month, year, hours, minutes, seconds or milliseconds. In order to manipulate the Date function you must create a new instance of the date within OpusScript using the new operator.

Syntax:

var dateName = new Date(parameters)

Remarks:

var - is used to create a new variable in which the date will be stored.

dateName – the name you want to give to the variable.

new – the operator that states you are creating a new instance of the object specified.

Date – the object you want to create a new instance of.

Parameters – this is optional. If no parameter is entered the new Date is set to the current date and time on your PC. You can enter the parameter as a number of milliseconds since Midnight 1/1/1970 or you can enter the parameter as a string, showing the date and time or just the date you want to set (see below).

Example 1: New date object with no parameters

In this example, the variable myDate creates a new object with no parameters. 2 and 3 below also create a date object but a specific date has been added. The date contained in the new object for Example 1 will be the current date and time on your computer

```
var myDate = new Date()
```

Example 2: New date set by entering milliseconds

In this example, the variable myDate creates a new date based on number of milliseconds you have entered. This is the number of milliseconds that have passed since Midnight 1/1/1970

```
var myDate = new Date(949278000000)
```

Note: The date and time will be set to 31 January 2000 00:20:00

Example 3: New date set by entering a string

In this example the date is set by entering a string showing the date, month, year followed by the time in hours, minutes and seconds. You must enter the string in the same syntax as the example

```
var myDate = new Date("25 December 2001, 00:00:00")
```

Example 4: Using the Date object properties

When you have created a new date object, the object contains a list of properties that allow you to get sections of the date and store them to a variable or alternatively, you can set sections of the date to a new value. The full list of properties is available in Date section of the Help file – see Date Overview for more information.

```
var myDate = new Date() var currHour = myDate.getHours()
```

In the example above, the variable currHour will contain the current hour e.g. the number 0 indicates midnight, 22 indicates 10pm and so on. In the example below, the setHours function is used to change the hours for the date object myTime, the new date is then stored in a variable showTime

```
var myTime = new Date() var currTime = myTime.setHours(3) var showTime =
myDate
```

Basic Objects - Overview

All objects that appear in the Objects tab of the Organiser on the left- hand side of the Opus Editor can be used in a script and can use the Basic Objects functions.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Basic objects:

Class Hierarchy:

Publications Graphical Objects PagesTimelineBrowser

Basic Objects

Functions:

FindChild Finds the child of this object with a particular

name

FindDescendant Finds the descendant of this object with a

particular name

GetChild Gets a child of this object by index

GetFirstChild Gets the first child of this object

GetName Gets the name of the object

GetNextChild Gets the child of this object that follows

another

GetNumberChildren Gets the number of children of this object

GetPage Gets the page this object is on

GetParent Gets the parent of this object

GetUniqueObjectID Gets a string that is unique to this object

GetPage

Syntax:

GetPage()

Return:

The Page the object is on.

Example:

```
var page = Object.GetPage()
```

GetParent

Syntax:

GetParent()

Return:

The parent of this object. No parameters required.

Remarks:

The parent of an object is the object one level above it in the Organiser tree. For example, the parent of any chapter is the publication.

Note: The publication has no parent and so GetParent() returns NULL for the publication.

Example:

For a Button object named myButton, use the syntax:

```
myButton.GetParent()
```

GetFirstChild

Syntax:

GetFirstChild()

Return:

The first child of this object. No parameters required.

Remarks:

If there are no children of this object then the Return Value is NULL.

The children of an object are the objects one level below it in the organiser tree. For example, the Chapters are the children of the Publication.

Note: All objects on the page are returned in the reverse order to that shown in the Organiser. In other words, the bottom most child is the first child retrieved in the Return Value.

Example:

For a Button object named myButton, use the syntax:

```
myButton.GetFirstChild()
```

GetNextChild

Syntax:

GetNextChild(Before)

Return:

The child of the specified object that follows Before.

Parameters:

Before - A child of this object. This parameter is required.

Remarks:

If Before is not a child of the object specified, or it is the last child in the object then the Return Value is NULL.

Note: All objects on the page are returned in the reverse order to that shown in the Organiser. In other words, the bottom most child is the first child retrieved in the Return Value.

Example:

For a Frame object named frame1 containing two Button objects named myButton and myOtherButton, use either of the following syntax in a Script Action of the frame:

```
frame1.GetNextChild(myButton) // or this.GetNextChild(myButton)
```

GetNumberChildren

Syntax:

GetNumberChildren()

Return:

The number of children this object has. No parameters required.

Remarks:

The count of children only includes Objects that are accessible from the script. Some objects that appear in the Organiser are not available in the script e.g. Script Objects.

Example:

For a Frame object named Frame1, to find the number of objects within the frame, use the following syntax:

```
var numOfChildren = Frame1.GetNumberChildren()
```

GetChild

Syntax:

GetChild(Index)

Return:

The name of the specified child of the specified Object.

Parameters:

Index - The index position of the required child. Index is zero-based indexed. This parameter is required.

Remarks:

This function returns the name of the specified Child.

Note: All objects on the page are returned in the reverse order to that shown in the Organiser. In other words, the bottom most child is the first child retrieved in the Return Value.

Example:

For a Frame object named Frame1, to find the first child in the frame, use the following syntax:

```
var myFirstChild = Frame1.GetChild(0)
```

FindChild

Syntax:

FindChild(Name)

Return:

The name of the child of the specified Object. If the child does not exist in the specified object, the Return Value is NULL.

Parameters:

Name - The name of the child. Name must be surrounded by quote marks. This parameter is required.

Remarks:

The name of the child used is the original name of the object, as shown in the Organiser. You should not replace spaces with underscores as you do when referring to an object directly e.g. for Button 1, use the name Button 1 NOT Button_1.

Example:

For a Frame object named Frame1, containing a Button object named myButton, use the following syntax:

```
var childFound = Frame1.FindChild("myButton") // or var childFound =
this.FindChild("myButton")
```

FindDescendant

Syntax:

FindDescendant(Name)

Return:

The object with the given name that is a child of the specified Object, or one of its children, or childrens children etc. The Return Value is NULL if no such object exists.

Parameters:

Name - The name of the child. This parameter is required.

Example:

For a Frame object named Frame1, which contains two descendants called Button1 and Button2, use the following syntax:

```
var descendantFound = Frame1.FindDescendant("Button1")
```

GetName

Syntax:

GetName()

Return:

The name of the specified object as it appears in the Organiser.

Remarks:

No parameters required.

Example:

```
var myName = this.GetName()
```

GetUniqueObjectID

Syntax:

GetUniqueObjectID()

Return:

A string that is a unique refrence to the specified Object.

Remarks:

No parameters required. The string returned is of the form ObjectXXXXXXXXXXXXXXXX where X are digits, or the letters A- F e.g. Object00_0001_00000004.

Example:

```
var myID = this.GetUniqueObjectID()
```

Polygon - Overview

The Polygon OpusScript functions are unique to Draw objects. When you create a Draw object with the Vector tool, it may be made up of a single line or a shape or a number of shapes and lines. Each drawn objects is known as a polygon.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Vector objects:

Class Hierarchy:

Polygon

Functions:

SetFillColour Set the fill colour of a polygon in a Draw object

SetLineColour Set the line colour of a polygon in a Draw object

SetFillColour

Syntax:

SetFillColour(Colour, Opacity)

Parameters:

Colour – the colour for the fill colour of the specified polygon. Colour can either one of:

• a string One of "white", "black", "red", "green", "blue", "yellow", "cyan" or "magenta".

• a string containing a hex value (i.e. a six digit number as used in HTML) e.g. "#45B93B" – note the string must begin with the hash symbol (i.e. #).

• a single RGB value. The RGB value can be calculated using the RGB function.

• three values for the amount of red, green and blue e.g. 255,0,0.

• Using –1 will result in no fill at all.

This parameter is required.

Opacity – the level of opacity of the colour from 0 – 100%

Remarks:

This function will set the fill colour of a specified polygon to the new colour entered in Colour.

Note: This function will only change the fill colour of a polygon created with the Vector tool or a vector that has been imported into Opus.

Example 1: Identify the polygon to change

The first Vector object you draw on a page is automatically given the name Vector 1. This Vector object will contain the first line or shape you drew and it will be named Polygon. When referencing an object with the SetFillColour function you must identify the polygon name and not the vector object. In other words, in this example, to change the fill colour of Vector 1, you must specify Polygon

```
Polygon.SetFillColour("green")
```

Note: Like any other object in Opus, you can obviously change the name of the vector and polygon to any name you choose. Use the GetChild function to get the name of a polygon in a vector object with multiple polygons.

Example 2: Change fill colour by Hex value

To change the fill colour of a polygon named Poygon using a Hex value, use the following syntax:

```
Polygon.SetFillColour("#7F5D9A") // a purplish colour
```

Example 3: Change fill colour by RGB function

To change the fill colour of a polygon named Polygon using the RGBfunction, use the following syntax:

```
Polygon.SetFillColour(RGB(200,98,33)) // or var newColour = RGB(200,98,33)
Polygon.SetFillColour(newColour)
```

Example 4: Change fill colour by specifying amount of Red, Green and Blue

To change the fill colour of a polygon named Polygon by the amount of red, green and blue, use the following syntax:

```
Polygon.SetFillColour(240,120,60) // or var red = 240 var green = 120 var blue = 60 Polygon.SetFillColour(red, green, blue)
```

SetLineColour

Syntax:

SetLineColour(Colour)

Parameters:

Colour – the colour for the fill colour of the specified polygon. Colour can either one of:

• a string One of "white", "black", "red", "green", "blue", "yellow", "cyan" or "magenta".

• a string containing a hex value (i.e. a six digit number as used in HTML) e.g. "#45B93B" – note the string must begin with the hash symbol (i.e. #).

• a single RGB value. The RGB value can be calculated using the RGBfunction.

• three values for the amount of red, green and blue e.g. 255,0,0.

This parameter is required.

Remarks:

This function will set the line colour of a specified polygon to the new colour entered in Colour.

Note: This function will only change the line colour of a polygon created with the Vector tool or a vector that has been imported into Opus.

Example 1: Identify the polygon to change

The first Vector object you draw on a page is automatically given the name Vector 1. This Vector object will contain the first line or shape you drew and it will be named Polygon. When referencing an object with the SetLineColour function you must identify the polygon name and not the vector object. In other words, in this example, to change the line colour of Vector 1, you must specify Polygon

```
Polygon.SetLineColour("green")
```

Note: Like any other object in Opus, you can obviously change the name of the vector and polygon to any name you choose.

Example 2: Change line colour by Hex value

To change the line colour of a polygon named Polygon using a Hex value, use the following syntax:

```
Polygon.SetLineColour("#7F5D9A") // a purplish colour
```

Example 3: Change line colour by RGB function

To change the line colour of a polygon named Polygon using the RGB function, use the following syntax:

```
Polygon.SetLineColour(RGB(200,98,33)) // or var newColour = RGB(200,98,33)
Polygon.SetLineColour(newColour)
```

Example 4: Change line colour by specifying amount of Red, Green and Blue

To change the line colour of a polygon named Polygon by the amount of red, green and blue, use the following syntax:

```
Polygon.SetLineColour(240,120,60) // or var red = 240 var green = 120 var
blue = 60 Polygon.SetLineColour(red, green, blue)
```

Browsers - Overview

The Browser OpusScript functions are unique to Browser objects in a publication.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Browsers

Basic Objects

Functions:

Back Goes back one page in the browser history

Forward Goes forward one page in the browser history

Home Goes to the Home page of the specified browser

Object

Navigate Open a specified URL in specified browser Object

Print Print the contents of the browser Object.

Refresh Redisplays the current page in the specified

browser Object

Stop Stops the specified browser Object

Back

Syntax:

Back()

Remarks:

This function will go back one page in the Browser's history of the specified Browser object.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Back()
```

Forward

Syntax:

Forward()

Remarks:

This function will go forward one page in the Browser's history of the specified Browser object.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Forward()
```

Home

Syntax:

Home()

Remarks:

This function will go to the Home page of the specified Browser object.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Home()
```

Navigate

Syntax:

Navigate(URL)

Parameters:

URL – enter the URL of the page you want to open in the specified browser. URL can either be

the full URL of a web site or a variable name that contains a URL. This parameter is required.

Remarks:

This function will open the requested URL in the specified Browser object.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Navigate("www.digitalworkshop.com") // or var link =
"www.digitalworkshop.com" myBrowser.Navigate(link)
```

Print

Syntax:

Print(ShowDialog)

Parameters:

ShowDialog – show the printer's settings dialog. ShowDialog can be set to true or false. If true, the printer's settings dialog is shown. If false, the dialog is not shown. This parameter is optional and the default is false.

Remarks:

This function print the contents of the Browser/DocView using the embedded program itself.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Print(true)
```

In this example, the printer's settings dialog will appear.

Refresh

Syntax:

Refresh()

Remarks:

This function will re-display (i.e. refresh) the current page displayed in the Browser object.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Refresh()
```

Stop

Syntax:

Stop()

Remarks:

This function will stop the specified Browser object.

Example:

For a Browser object named myBrowser, use the following syntax:

```
myBrowser.Stop()
```

Buttons - Overview

The Button functions allow you to use a script to assess the state of a Button object in your publication, in particular the state of a push button. For example, a push button can either be Up or Down, you can either use the GetState function to find out the current state of a button or use the SetState function to make the push button Up or Down. Once you have used these functions, you can use Graphical Object functions, such as, the Show function to have other objects on a page do something depending on the state of the specified push button.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Buttons

Basic Objects

Functions:

GetState Returns the current state of a push button

SetState Sets the current state of a push button

Graphical Objects

GetState

Syntax:

GetState()

Return:

true if the object state is currently down. Otherwise false if the button state is up.

Remarks:

This function returns the current state of the specified push Button object. No parameters required.

Example:

```
var butStatus = myButton.GetState() if (butStatus == true) {
   myButton.SetState(false) }
```

In the example above, the current state of the Button object named myButton is stored in the variable butStatus. The SetState function is used in an if statement to change the object state to Up if it is currently set to Down.

SetState

Syntax:

SetState(State)

Parameters:

State – Set State to true if the button state should be Down. Or set State to false if the button state should be Up. This parameter is required.

Remarks:

This function sets the state of the specified push Button object. If the Down status or Up status of a button has different button colours, images, effects or borders, these will be activated when this function is executed.

Example:

For a Button object named myButton, to set the button status to Down, use the following syntax:

```
myButton.SetState(true)
```

Clock - Overview

The Clock OpusScript functions are unique to clocks created in a script

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Clock objects:

Class Hierarchy:

Clock

Functions:

CreateClock Creates a new clock Object

Continue Continue a paused clock Object

GetClock Returns a specified clock Object

GetHours Returns number of hours for a clock Object

GetMinutes Returns number of minutes for a clock Object

GetSeconds Returns number of seconds for a clock Object

Pause Pause a clock Object

Start Start a clock Object

Stop Stop a clock Object

CreateClock

Syntax:

CreateClock(Name, Variable, Format)

Parameters:

Name – the name of the Clock object in OpusScript. Name must be surrounded in quote marks. This parameter is required.

Variable – the name of the variable that will display the Clock object on a page within the publication. Variable must be surrounded in quote marks. This parameter is required.

Format – the formatting of the clock on the page. A Clock object measures hours, minutes and seconds. Hours are formatted either as [h] or [hh]; minutes are formatted as [m] or [mm]; and seconds are formatted as [s] or [ss]. Other characters can be placed within the format. This parameter is optional and the default format is: [hh]:[mm]:[ss].

Remarks:

This function will create a Clock object in OpusScript. The clock can be displayed on a page inside the named Variable. The clock will not start until the clock Start function is executed.

Example:

```
var myClock = CreateClock("ScriptClock","clock_var") myClock.Start() // will start the clock.
```

GetClock

Syntax:

GetClock(Name)

Parameters:

Name – the name of a Clock object created in OpusScript. Name must be surrounded in quote marks. This parameter is required.

Remarks:

This function will get a Clock object previously created in OpusScript.

Example:

```
var getMyClock = GetClock("ScriptClock")
```

Start

Syntax:

Start()

Remarks:

This function will start a Clock object which has been created using the CreateClock function.

Example:

For a Clock object named myClock, use the following syntax:

```
myClock.Start()
```

Stop

Syntax:

Stop()

Remarks:

This function will stop a Clock object at its current time. When this function is used, the clock can only be re-started using the Start function, you cannot use the clock's Pause or Continue functions.

Example:

For a Clock object named myClock, use the following syntax:

```
myClock.Stop()
```

Pause

Syntax:

Pause()

Remarks:

This function will pause a Clock object at its current time. When this function is used, the clock can be re-started using the Continue function. To re-start the clock from zero, use the clock Start function.

Example:

For a Clock object named myClock, use the following syntax:

```
myClock.Pause()
```

Continue

Syntax:

Continue()

Remarks:

This function will continue a Clock object from its current paused time. To re-start the clock from zero, use the clock Start function.

Example:

For a Clock object named myClock, use the following syntax:

```
myClock.Continue()
```

GetHours

Syntax:

GetHours()

Return:

The number of hours the specified Clock object has been playing.

Remarks:

This function will return the number of hours a Clock object has been playing.

Example:

For a Clock object named myClock, use the following syntax:

```
var clockHours = myClock.GetHours()
```

GetMinutes

Syntax:

GetMinutes()

Return:

The number of minutes the specified Clock object has been playing.

Remarks:

This function will return the number of minutes a Clock object has been playing.

Example:

For a Clock object named myClock, use the following syntax:

```
var clockMinutes = myClock.GetMinutes()
```

GetSeconds

Syntax:

GetSeconds()

Return:

The number of seconds the specified Clock object has been playing.

Remarks:

This function will return the number of seconds a Clock object has been playing.

Example:

For a Clock object named myClock, use the following syntax:

```
var clockSeconds = myClock.GetSeconds()
```

File - Overview

The File functions allow you to open a file in a script and then read and write lines to the file. These OpusScript functions are equivalent to using the Storage actions in the Storage menu of the Actions dialog. To use these functions, you must first use the OpenFile function to create a new File object in the script.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with File objects:

Class Hierarchy:

File

Functions:

Close Close an opened file

EndOfFile Check for the end of file marker in the specified

file

GotoFirstLine Goes to first line of an opened file

GotoNextLine Goes to next line of an opened file

OpenFile Opens a file and returns a new file Object

Read Read a file that has been opened

ReadField Reads a field in a file that has been opened

ReadLine Reads first line of an opened file

toString Returns the type of object as a string

Write Write a string to an opened file

WriteField Write a string to a field in an opened file

WriteLine Write a string to a line in an opened file

OpenFile

Syntax:

OpenFile(Filename, Overwrite, ReadOnly)

Return:

A new object, which has a variety of properties, such as, Read and Write. The properties can be manipulated using OpusScript File functions.

Parameters:

Filename – enter the full pathname of a text file. Filename should be the full pathname of the text file or the alias name given to the file in the Additional Resources tab of the Page Properties or Publication Properties dialog. This parameter is required.

Note: Pathnamesnormally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

If Filename is not found, then the file is created automatically.

Overwrite – overwrite the file specified. Overwrite can be true or false. If true it overwrites the current file. If false, the data is written to the end of the file. This parameter is optional and the default is false.

ReadOnly – set the read-only mode of the file. ReadOnly can be true or false. If true the file will be opened in read-only mode (i.e. no changes can be made to the file) – this also means the Overwrite parameter cannot be true if ReadOnly is true. If false, the file can be changed. This parameter is optional and the default is false.

Remarks:

This function will create a new OpusScript object that will allow you to manipulate the contents of the specified text file using File functions. If the specified file is not found, this function will create the file for you.

Example:

```
var textObj = OpenFile("d:\\myText.txt")
```

Read

Syntax:

Read(Encrypt, Key)

Return:

The read selection of the file.

Parameters:

Encrypt – set the encryption on or off. Encrypt is either true or false. If Encrypt is true, the file is encrypted. If Encrypt is false, the file is not encrypted. This parameter is optional and the default is false.

Key – a string that is the encryption key. This parameter is optional.

Remarks:

This function will read a file that has been opened with a OpenFile function.

Example:

```
var textObj = OpenFile("d:\\myText.txt") textObj.Read(false)
```

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

ReadLine

Syntax:

ReadLine(Encrypt, Key)

Return:

The first line of the file.

Parameters:

Encrypt – set the encryption on or off. Encrypt is either true or false. If Encrypt is true, the file is encrypted. If Encrypt is false, the file is not encrypted. This parameter is optional and the default is false.

Key – a string that is the encryption key. This parameter is optional.

Remarks:

This function will read the first line of a file that has been opened with a Open File function.

Example:

```
var textObj = OpenFile("d:\\myText.txt") textObj.ReadLine(false)
```

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

ReadField

Syntax:

ReadField(Encrypt, Key)

Return:

All the text up to the first comma. If the text is not comma separated, then the full text is returned.

Parameters:

Encrypt – set the encryption on or off. Encrypt is either true or false. If Encrypt is true, the file is encrypted. If Encrypt is false, the file is not encrypted. This parameter is optional and the default

is false.

Key – a string that is the encryption key. This parameter is optional.

Remarks:

This function will read a comma-separated file that has been opened with an Open File function. Each time the function is called, the next field (i.e. text separated by commas) will be displayed.

Example:

```
var textObj = OpenFile("c:\\myText.txt") textObj.ReadField(false)
```

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

Write

Syntax:

Write(String, Encrypt, Key)

Parameters:

String – a string containing the text you want to write to the File object. The string must be surrounded by quote marks. This parameter is required.

Encrypt – set the encryption on or off. Encrypt is either true or false. If Encrypt is true, the file is encrypted. If Encrypt is false, the file is not encrypted. This parameter is optional and the default is false.

Key – a string that is the encryption key. This parameter is optional.

Remarks:

This function will write a string to a line in a file that has been opened with a Open Filefunction.

Example:

```
var textObj = OpenFile("c:\\myText.txt") textObj.Write("Hello World")
```

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

WriteLine

Syntax:

WriteLine(String, Quote, Encrypt, Key)

Parameters:

String – a string containing the text you want to write to the File object. The string must be surrounded by quote marks. This parameter is required.

Quote – surround the String with quotation marks. Quote is either true or false. If Quote is false,

the string is not surrounded by quote marks. If Quote is true, the string is surrounded by quote marks. This parameter is optional and the default is false.

Encrypt – set the encryption on or off. Encrypt is either true or false. If Encrypt is true, the file is encrypted. If Encrypt is false, the file is not encrypted. This parameter is optional and the default is false.

Key – a string that is the encryption key. This parameter is optional.

Remarks:

This function will write a string to a line in a file that has been opened with an Open File function.

Example:

```
var textObj = OpenFile("c:\\myText.txt") textObj.WriteLine("Hello World")
```

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

WriteField

Syntax:

WriteField(String, Quote, Encrypt, Key)

Parameters:

String – a string containing the text you want to write to the File object. The string must be surrounded by quote marks. This parameter is required.

Quote – surround the String with quotation marks. Quote is either true or false. If Quote is false, the string is not surrounded by quote marks. If Quote is true, the string is surrounded by quote marks. This parameter is optional and the default is true.

Encrypt – set the encryption on or off. Encrypt is either true or false. If Encrypt is true, the file is encrypted. If Encrypt is false, the file is not encrypted. This parameter is optional and the default is false.

Key – a string that is the encryption key. This parameter is optional.

Remarks:

This function will append the contents of the string to the end of the file that has been opened with an Open File function. When the string is entered in the file, the string is prefixed with a comma and the string is optionally surrounded by quote marks e.g. ,"new string". Normally, this function is used when adding data to a comma separated file.

Example:

```
var textObj = OpenFile("c:\\myText.txt") textObj.WriteField("Hello World")
```

Note:

Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

GotoFirstLine

Syntax:

GotoFirstLine()

Remarks:

This function will go to the first line of a file that has been opened with an Open File function.

```
Example: var textObj = OpenFile("d:\\myText.txt") textObj.GotoFirstLine()
textObj.WriteLine("Hello World")
```

Note:

Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

GotoNextLine

Syntax:

GotoNextLine()

Remarks:

This function will go to the next line of a file that has been opened with an Open File function.

```
Example: var textObj = OpenFile("d:\\myText.txt") textObj.GotoFirstLine()
textObj.WriteLine("Hello World") textObj.GotoNextLine()
```

Note:

Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

EndOfFile

Syntax:

EndOfFile()

Return:

true or false. If true, the end of file was reached. If false, the end of file was not reached.

Remarks:

This function will check for the end of file marker in the specified file.

Example:

For a File object named textObj, use the following syntax:

```
var textObj = OpenFile("d:\\myText.txt") while (textObj.EndOfFile() == false)
{
      textObj.ReadLine() }
```

Note:

Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

Close

Syntax:

Close()

Remarks:

This function will close a file that has been opened with an Open File function.

```
Example: var textObj = OpenFile("d:\\myText.txt") textObj.WriteLine("Hello World") textObj.Close()
```

Note:

Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

toString

Syntax:

toString()

Return:

A string specifying the object type for the script name you have requested.

Remarks:

This function will return a string containing an object type.

Example:

In the example below, the variable task plays a sound file. The toString function checks to see if task contains a sound object and if it does it will show the Image object named soundImage

```
task = PlaySound("E:\\Bleep.WAV") showImage = task.toString() if (showImage == "[Sound]") {
  soundImage.Show() }
```

Global - Overview

The Global functions can be called without a reference to an object in a publication. In Opus Pro, the publication and the page are Global objects and the functions here are properties of those objects.

In addition to these functions, there are also Global String functions that perform many simple string and conversion operations.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Global objects:

Class Hierarchy:

Global

Functions:

ChangeDisplayMode Change the user's Display Mode (i.e. screen resolution) when running the publication

CopyFile Copy a file to a new location of the

user's machine

CountLines Count the number of lines in a text

object or string/variable

CountWords Count the number of words in a text

object or string/variable

CreateCDPlayer Create a new CD player Object

Debug.trace Debugging tool that uses the Script

Console

Debug.tracePane Debugging tool that creates a new pane

in the Script Console

DeleteFile Delete a file from the user's machine

DeleteRegistryValue Delete a name/value pair for the system

registry

ExitPublication Exit the publication, via an Exit page (if

any)

FileExists Test if a file exists in a given location

Fork Allows lines of code to be executed

simultaneously

GetINIFileData Return the value of a specified key in a

specified section of an INI file
GetINISectionData Return the contents of a section of an
INI file as a string
GetJoystickState Return the current state of the first
system joystick
GetLastPage Get the last page in the publication
GetPage Get a page by name
GetPageDownloadPercent Get the percentage complete of the page
downloaded
GetPageDownloadPosition Get the position of the page downloaded
GetPageDownloadTotal Get the total size of the page
Downloaded
GetType Get the object's Object Type e.g. button, image etc.
GotoBackwardPage Go to the previous page in the publication
GotoCurrentPage Go to the current page in the page
history
GotoForwardPage Go to the next page in the publication
GotoNextPage Go to the next page in the page history
GotoNextRandomPage Go to a random page within the chapter without repeating
GotoPage Go to a given page
GotoPreviousPage Go to the previous page in the page
history
GotoRandomPage Go to a random page within the chapter
HtmlHelp Open a Compile HTML help file.
InternetGetData Gets information from a remote server
InternetPostData Post information to a remote server
IsKeyPressed Test if a specific key is being pressed
IsMousePressed Test if a specified special key is being pressed
IsPreview Test it the current player is the Opus
edior preview
LaunchFile Open an external file
LaunchSearch Open the Search dialog box
LaunchURL Open a URL

OpenSearch Returns a search object containing the publication search words

PlayCDTrack Play a track from a CD

PlaySystemSound Play a system sound from the users computer

PrintFile Print the specified file

PrintObject Print the specified object

PrintPage Print the specified page or page Object

ReadRegistryKey Return the value of a specified key in the system registry

ReallyExitPublication Exit the publication immediately

RGB Return a number indicating a RGB colour value

SendEmail Send an email (requires Outlook)

WinHelp Open a standard Windows help file

WriteRegistryValue Write a name/value pair to the system registry

ChangeDisplayMode

Syntax:

ChangeDisplayMode(DisplayIndex, DepthIndex)

Return:

true if the display mode is changed by this function and false if it is not.

Parameters:

DisplayIndex – the index position of the new display mode you want to set. DisplayIndex is an index containing possible screen resolutions (i.e. the display mode in which the user's screen can be set). The index appears in the following order, 0 = 640x480; 1 = 800x600; 2 = 1024x768, and so on. The index is a zero-based index (it begins with 0 not 1) This parameter is optional. The default value is –1, which means the display mode is set to the default for the machine.

DepthIndex – the colour depth you want to set for the end-user's computer. DepthIndex can be any of the following numbers: 15, 16, 24 or 36 (e.g. 16bpp). This parameter is optional. The default value is -1, which means the colour depth is set to the default for the machine.

Remarks:

This function allows you to change the display mode (i.e. screen resolution) and colour depth of the end-user's machine temporarily while the publication is run. At any time you can re-use the function to return the screen resolution or colour depth to their default value.

Example 1:

To change the display mode of the user's screen to 640x400 and leave the current colour depth at the default setting, use the following syntax:

```
ChangeDisplayMode(0)
```

Example 2:

You can check if this function does change the user's screen display by checking the Return Value. If the Return Value is true, the display mode was changed, if false it was not. In the example below, the mode is changed to 800x600.

```
var ModeChanged = ChangeDisplayMode(1) if (ModeChanged) {
      //when display mode changed run these statements } else {
      // when display mode NOT changed run these statements }
```

Note: The else function could contain a list of statements indicating what to do if the display mode was not changed, such as trying again with a different display mode or showing a warning message to the user.

GotoPage

Syntax:

GotoPage(Page)

Parameters:

Page – the name of the page to be displayed. Page can either be a string containing the page name or a variable containing the page name. This parameter is required.

Remarks:

This function will go to the specified page.

Example 1:

```
GotoPage("Page 1")
```

Example 2:

```
var pageName = "Page 2" GotoPage(pageName)
```

GotoForwardPage

Syntax:

GotoForwardPage()

Remarks:

This function will go to the next page in the publication. No parameters required.

Example:

```
GotoForwardPage()
```

GotoBackwardPage

Syntax:

GotoBackwardPage()

Remarks:

This function will go to the previous page in the publication. No parameters required.

Example:

```
GotoBackwardPage()
```

GotoNextPage

Syntax:

GotoNextPage()

Remarks:

This function will go to the next page in the page history. As a publication runs, Opus keeps a list, called the Page History, of all the pages visited. The GotoNextPage and GotoPreviousPage functions will move through this list. If more than one view is open at one time, such as the Layout panels or chapters in separate windows, then each view has its own Page History. No parameters required.

Example:

```
GotoNextPage()
```

GotoPreviousPage

Syntax:

GotoPreviousPage()

Remarks:

This function will go to the previous page in the page history. As a publication runs, Opus keeps a list, called the Page History, of all the pages visited. The GotoNextPage and GotoPreviousPage functions will move through this list. If more than one view is open at one time, such as the Layout panels or chapters in separate windows, then each view has its own Page History. No parameters required.

Example:

```
GotoPreviousPage()
```

GotoCurrentPage

Syntax:

GotoCurrentPage()

Remarks:

This function will go to the current page in the page history. This can be useful to reset a page back to its initial state. As a publication runs, Opus keeps a list, called the Page History, of all the pages visited. The GotoNextPage and GotoPreviousPage functions will move through this list. If more than one view is open at one time, such as the Layout panels or chapters in separate windows, then each view has its own Page History. No parameters required.

Example:

```
GotoCurrentPage()
```

GotoRandomPage

Syntax:

GotoRandomPage()

Remarks:

This function will go to a random page from this chapter. This page may be any page other than the current page being displayed. No parameters required.

Example:

```
GotoRandomPage()
```

GotoNextRandomPage

Syntax:

GotoNextRandomPage()

Remarks:

This function will go to a random page from this chapter, without repeating a page already displayed. The first time this function is called, it creates a list of all the pages in this chapter. Subsequent calls to this function will return the next page from the generated list. When all the pages in the generated list have been displayed, a new list is randomly generated and the process begins again. No parameters required.

Example:

```
GotoNextRandomPage()
```

ExitPublication

Syntax:

ExitPublication()

Remarks:

This function will exit the current publication via the Exit page. If an Exit page has been set for this publication, this function will close the current page and go to the Exit page. If this function is called on the Exit page itself, the publication will be closed. No parameters required.

Example:

```
ExitPublication()
```

ReallyExitPublication

Syntax:

ReallyExitPublication()

Remarks:

This function will exit the current publication ignoring any Exit Page. If an Exit page has been set for this publication, this function will immediately close the publication down, ignoring the Exit page. No parameters required.

Example:

```
ReallyExitPublication()
```

GetPage

Syntax:

GetPage(Name)

Return:

The page object specified in Name.

Parameters:

Name – The name of the page to return. Name is either the name of the page entered as a string, which should be surrounded in quote marks; or the index position of the page (the pages are based on a zero-based index, that is the first number is 0 not 1). This parameter is required.

Remarks:

This function gets a page and returns its page object. The page object can be used with other OpusScript functions, such as print Page. The parameter Name is used to specify the page either by its string name or by its index position.

Example 1:

To return a page object into a variable named page, use the following syntax:

```
var page = GetPage("page 1") // this gets the page object via the Name var
page = GetPage(0) // this gets the page via the Index and returns the first
// page in the publication
```

Example 2:

The returned page object can be used with OpusScript functions related to the page. For example, to print the first page in the publication, use the following syntax:

```
var firstPage = GetPage(0) PrintPage(firstPage,true)
```

GetLastPage

Syntax:

GetLastPage()

Return:

The page object of the last page in the publication.

Remarks:

This function returns a page object for the last page in the publication. No parameters required.

Example:

To return the page object into a variable named page for the last page in your publication, use the following syntax:

```
var lastPage = GetLastPage()
```

Example 2:

The returned page object can be used with OpusScript functions related to the page. For example, to print the last page in the publication, use the following syntax:

```
var lastPage = GetLastPage()
```

```
PrintPage(lastPage,true)
```

GetFirstView

Syntax:

GetFirstView()

Return:

The first view currently open.

Remarks:

This function will get the first view currently open. While the publication is running, Opus maintains a list of all the views that are open. The GetFirstView and GetNextView functions allow access to the list. However, the list is not guaranteed to be in any particular order. No parameters required.

Example:

```
var view = GetFirstView()
```

GetNextView

Syntax:

GetNextView(View)

Parameters:

View – View to get the next view after. This parameter is required.

Return:

The view that follows the given one in the list of currently open views.

Remarks:

This function will get the view that follows the given view in the list of currently open views. If there is no other view, NULL is returned. While the publication is running, Opus maintains a list of all the views that are open. The GetFirstView and GetNextView functions allow access to the list. However, the list is not guaranteed to be in any particular order.

Example:

```
var view = GetFirstView() while (view) {
Debug.trace( view.GetPage().GetName() ); view = GetNextView(view); }
```

GetPageDownoloadPercent

Syntax:

GetPageDownoloadPercent(Page)

Return:

The percentage of the page that has been downloaded.

Parameters:

Page – the name of the page for which you want to check download information. Page can be a page name surrounded by quotes or a variable containing a page name. This parameter is required.

Remarks:

This function will return the percentage of the specified page that has currently been downloaded.

Example:

```
GetPageDownloadPercent("Page 1")
```

GetPageDownloadPosition

Syntax:

GetPageDownloadPosition(Page)

Return:

The position of the page downloaded.

Parameters:

Page – the name of the page for which you want to check download information. Page can be a page name surrounded by quotes or a variable containing a page name. This parameter is required.

Remarks:

This function will return the position of the page being downloaded.

Example:

```
GetPageDownloadPosition("Page 1")
```

GetPageDownloadTotal

Syntax:

GetPageDownloadTotal(Page)

Return:

The total amount of the page downloaded.

Parameters:

Page – the name of the page for which you want to check download information. Page can be a page name surrounded by quotes or a variable containing a page name. This parameter is

required.

Remarks:

This function will return the total amount of the page downloaded.

Example:

```
GetPageDownloadTotal("Page 1")
```

IsKeyPressed

Syntax:

IsKeyPressed(Key1, Key2, ...)

Return:

true if a specified key is pressed down. Otherwise, false.

Parameters:

Key1, Key2, – a single character string representing a text key (e.g. a, b, c) or one of special keys listed below. More than one Key can be included within each function. Each Key should be separated by a comma. This parameter is required.

Special Keys:

One or more of the following strings: "Up", "Down", "Left" or "Right", representing the keyboard arrow keys. As well as one or more of the following: "Control", "Shift","Alt", "Backspace","Enter", "Caps","Pause", "Tab","Escape", "PageUp","PageDown", "End","Home", "Print","Insert", "Delete","Help", "LeftWin","RightWin", "Add","Multiply", "Subract", "Decimal", "Divide", "F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9", "F10", "F11", "F12", "F13", "F14", "F15", "F16", "F17", "F18", "F19", "F20", "F21", "F22", "F23", "F24" , "Numpad0", "Numpad1", "Numpad2", "Numpad3", "Numpad4", "Numpad5", "Numpad6", "Numpad7", "Numpad8", "Numpad9", "NumLock"

Remarks:

This function will return true when a specified key is being held down. This function is useful for programming keys to perform different actions in your script.

Example:

```
var score = 0 while (score <= 50) {
if (IsKeyPressed("Up")) {
      score = score + 1 } if (IsKeyPressed("Down")) {
      score = score – 1 } if (IsKeyPressed("b")) {
      break } }
```

IsMousePressed

Syntax:

IsMousePressed(Button1, Button2, ...)

Return:

true if a specified mouse button is pressed down. Otherwise, false.

Parameters:

Button1, Button2, – one of the following strings: "Left", "Right", "Middle", representing the left, right and middle buttons on a mouse. More than one Button can be included within each function. Each Button should be separated by a comma. This parameter is required.

Remarks:

This function will return true when a specified mouse button is being held down. This function is useful for programming keys to perform different actions in your script.

Example 1:

In the first example, the variable LButtonDown will contain the value true if the left mouse button is held down when the function was called, otherwise the value is false.

```
var LButtonDown = IsMousePressed("Left") // test if the left mouse button is down
```

Example 2:

In the second example, the variable MouseDown will contain the value true if the left and/or right mouse button is held down when the function was called, otherwise the value is false

```
var MouseDown = IsMousePressed("Left","Right") // test if the left and/or right mouse buttons are down
```

GetType

Syntax:

GetType()

Return:

A string containing the Object Type that the specified object belongs to e.g. "Page", "Button", "Image" or "Vector".

Remarks:

This function will return the specified object's Object Type. Possible values are

Non-Graphical Objects

"Publication" "Chapter"

"Page" "Timeline" "Script" "ObjectList"

Graphical Objects

"Frame" "Button" "Image" "Text" "Video" "Slideshow" "Vector" "Path" "Scrollbar" "Browser"

"Frameset" "Rollover" "ListBox"

Vector Objects

"Polygon"

Any unknown object type will be returned as "Unknown"

Example 1:

```
var objType = GetType() // will return "Page"
```

Example 2:

```
var objType = welcome.GetType() // will return the type for the object named
'welcome' e.g. "Button" if (welcome.GetType() == "Button") {
    welcome.SetState(true) }
```

PrintPage

Syntax:

PrintPage(PageName, PrintDialog, CancelDialog)

Parameters:

PageName – the name of the page to print. The page can be a specific page surrounded by quotes or a Page object. This parameter is optional, if no page is specified the current page is printed.

PrintDialog – display the printer poperties dialog box before printing. PrintDialog can be true or false. If true the print dialog box will appear. If false, the print dialog will not appear. This parameter is optional and the default is false.

CancelDialog - display the Cancel dialog box while the page is printing. CancelDialog can be true or false. If true, the cancel dialog will appear. If false, the cancel dialog will not appear. This parameter is optional and the default is true.

Remarks:

This function will print the specified page or page object. If no page is specified, the current page is printed.

Example 1:

```
PrintPage() // prints without dialog boxes displaying
```

Example 2:

```
PrintPage("Page 3", true, false) // prints page 3 with dialog boxes
```

PrintFile

Syntax:

PrintFile(Filename, PrintDialog, CancelDialog)

Parameters:

Filename – the name of the file to print. This parameter is required.

PrintDialog – display the printer poperties dialog box before printing. PrintDialog can be true or false. If true the print dialog box will appear. If false, the print dialog will not appear. This parameter is optional and the default is false.

CancelDialog - display the cancel dialog box while the page is printing. CancelDialog can be true or false. If true, the cancel dialog will appear. If false, the cancel dialog will not appear. This parameter is optional and the default is true.

Remarks:

This function will print the specified file.

Example 1:

```
PrintFile("d:\\moreinfo.doc") // prints without dialog boxes displaying
```

Example 2:

```
PrintFile("d:\\moreinfor.doc,true,false) // prints file with dialog boxes
```

PrintObject

Syntax:

PrintObject(PrintDialog, CancelDialog)

Parameters:

PrintDialog – display the printer poperties dialog box before printing. PrintDialog can be true or false. If true the print dialog box will appear. If false, the print dialog will not appear. This parameter is optional and the default is false.

CancelDialog - display the cancel dialog box while the page is printing. CancelDialog can be true or false. If true, the cancel dialog will appear. If false, the cancel dialog will not appear. This parameter is optional and the default is true.

Remarks:

This function will print the object it is called on.

Example:

For an Image object named certificate, use the following syntax:

```
certificate.PrintObject() // prints without dialog boxes displaying
```

DeleteFile

Syntax:

DeleteFile(Filename, MsgBox)

DeleteFile(Filename, MsgBox, Recycle)

Return:

true if the specified file is deleted and false if it is not.

Parameters:

Filename – enter the full pathname of a text file. Filename should be the full pathname of the text file or the alias name given to the file in the Additional Resources tab of the Page Properties or Publication Properties dialog. This parameter is required.

Note:

Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

MsgBox – display a warning box asking the user if they are sure they want to delete the file. MsgBox can be true or false. If true the warning box will appear on screen. If false, the warning box will not appear. This parameter is optional and the default is false.

Recycle – send the deleted file to the Recycle Bin. Recycle can be true or false. If true the file is sent to the Recycle Bin. If false, the file is permanently deleted. This parameter is optional and the default is false.

Note: When using the Recycle Bin, you must use a complete path for the filename (i.e. include drive, directories and filename) otherwise the delete will occur without using the Recycle Bin. It is always a good idea to use full paths to can be certain you are deleting the file you think you are.

Remarks:

This function will delete a specified file from the user's machine.

This function will only work in the full player - not the screensaver, web plug-in or Flex.

Example 1:

To delete a text file named UserInfo.txt and show a warning box on screen, use the following syntax:

```
DeleteFile("C:\\Users\\UserInfo.txt",true)
```

Example 2:

To check if the file has been deleted, check that the Return Value is true. If the Return Value is false then the file was not deleted and you can run the list of statements under the else function

```
FileToDelete = "C:\\Users\\UserInfo.txt" if (DeleteFile(FileToDelete,true)) {
      // if file is deleted run these statements } else {
      // if file is NOT deleted run these statements }
```

CopyFile

Syntax:

CopyFile(Source, Destination, MsgBox)

Return:

true if the specified file is copied and false if it is not.

Parameters:

Source – the full pathname of the file to be copied file. This parameter is required.

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

Destination – filename/path of the destination. This can be

• a simple filename - to make a copy of the source file in the same directory.

• a full path with no filename - to copy the file with the same name to another directory.

• a full path with filename - to copy the file to another directory with a new name.

Note: If the given destination directory does not exist it will be created automatically.

MsgBox – if the destination file exists, display a warning box asking the user if they are sure they want to overwrite the file. MsgBox can be true or false. If true the warning box will appear on screen. If false, the warning box will not appear and the file will be overwritten. This parameter is optional and the default is false.

Remarks:

This function will copy a specified file to a new location on the user's machine.

This function will only work in the full player - not the screensaver, web plug-in or Flex.

Example:

To copy a file from the publication directory to a temporary location:

```
CopyFile( SYSTEM_PUBLICATION_DIR + "\\test.txt", SYSTEM_TEMP_DIR )
```

FileExists

Syntax:

FileExists(Filename)

Return:

A number indicating the state of the file:

Value Meaning

0 the file does not exist

1 the file exists

2 the file exists but is read- only

Parameters:

Filename – the full pathname of the file to test for. This parameter is required.

Note: Pathnames normally contain backslashes e.g. d:\welcome.txt, all single backslashes should be entered as double backslashes i.e. d:\\welcome.txt.

Remarks:

This function will test if a specified fileexists at the given location.

This function will only work in the full player - not the screensaver, web plug-in or Flex.

Example:

To test if a file exists before attempting to copy a file from the publication directory to a temporary location:

```
if (FileExists( SYSTEM_PUBLICATION_DIR + "\\test.txt" )
{
      CopyFile( SYSTEM_PUBLICATION_DIR + "\\test.txt", SYSTEM_TEMP_DIR ) }
```

InternetPostData

Syntax:

InternetPostData(URL, Data, ReturnObject)

Parameters:

URL – the Uniform Resource Locator. The URL must indicate the script that will evaluate the Data sent back, such as, a cgi-script. URL can be a string or a variable containing a string. A string must be surrounded by quote marks. This parameter is required.

Data – the data to be sent and returned from the server. Data can be a string or an Object. A string can be a literal string surrounded by quote marks or a variable containing a string. An object can contain any number of members. This parameter is required.

ReturnObject – this parameter is either true or false. If true, the data is returned as an Object. If false, the data is returned as a string. This parameter is optional and the default is false.

Remarks:

This function will send and receive information from a remote server.

Example 1:

```
var url = "http://www.example.com/cgi- bin/checkdata.pl" // set up URL. var
out = InternetPostData(url,"data=Opus") // will return a string: "data=Opus"
out = InternetPostData(url,"data=Opus",true) // will return an object with
member 'data' set to "Opus".
```

Example 2:

```
// To send an Object, first set up the Object: var obj = new Object()
obj.Name = "John" obj.Age = 21 obj.ShoeSize = 8 out = InternetPostData(url,
obj, true) // will return an object with Name, Age and ShoeSize members set.
```

InternetGetData

Syntax:

InternetGetData(URL)

Parameters:

URL – the Uniform Resource Locator. The URL must indicate the script that will evaluate the data sent back, such as, a cgi-script. URL can be a string or a variablecontaining a string. A string must be surrounded by quote marks. This parameter is required.

Remarks:

This function will return information from a remote server.

Example:

For a Button object named getData to retrieve data from a server, use the following syntax:

```
var newData = InternetGetData("www.example.com/cgi- bin/checkdata.pl")
```

OpenSearch

Syntax:

OpenSearch()

Return:

A new search object containing a list of search words in the publication.

Note: There are four functions relating to the search Object: GetFirstKeyword, GetNextKeyword, GetFirstPage, GetNextPage.

Remarks:

This function will return a search Object, used for the publication search. To find keywords in the search list or find the page object that contains the keyword, you must first create a search object using the OpenSearch function.

Note: The Use publication search database option in the General tab of the Publication Properties dialog must be ticked (i.e. turned on) in order for the OpenSearch function to work when your publication is published.

Example:

To open a new search object and get the first keyword in the list of entries in the search, use the following syntax:

```
var Search = OpenSearch() var Keyword = Search.GetFirstKeyword()
```

CreateCDPlayer

Syntax:

CreateCDPlayer(Track)

Parameters:

Track – the number of the track on the CD that you want to play. Track can either be a number or a variablecontaining a number. This parameter is required.

Remarks:

This function is used to create a new object that is used to reference a CD audio within OpusScript.

Example:

```
var myAudioCD = CreateCDPlayer() myAudioCD.Play(3)
```

PlayCDTrack

Syntax:

PlayCDTrack(Track, Times, Volume, Start, Finish, FadeIn, FadeOut, Stop, Wait)

Parameters:

Track – The number of the track to play from the CD. This parameter is required.

Times – The number of times to play the track. To play the track continuously enter –1, otherwise enter a positive number. This parameter is optional. The default value is 1.

Volume – The percentage volume at which to play the track. Volume must be a number between 0 and 100. This parameter is optional. The default value is 100.

Start – The start position, relative to the beginning of the track, in seconds from which to play the track. Start must be a positive number. This parameter is optional. The default is 0, which is the beginning of the track.

Finish – The finish position, relative to the beginning of the track, in seconds from which to finish the track. Finish must be a positive number. This parameter is optional. The default is -1, which is the end of the track.

FadeIn – a length in seconds in which to fade in the track from the beginning. FadeIn must be a positive number. This parameter is optional. The default value is 0 i.e. no fade in.

FadeOut – a length in seconds in which to fade out the track from the end. FadeOut must be a positive number. This parameter is optional. The default value is 0 i.e. no fade out.

Stop – Sets if the track should stop when the page changes to another page. Stop can either be true or false. If Stop is true, the page will stop when the page changes. If Stop is false, the track will continue when the page changes. This parameter is optional. The default is true.

Wait – set to true or false. If Wait is true, the next line of code in the Script will not play until the track is finished. Otherwise, if Wait is false, the next line of code in the Script will run immediately. This parameter is optional and the default is false.

Remarks:

This function will play a specified track from a CD with the optional parameter settings.

Example:

```
var myTrack = PlayCDTrack(3)
```

Debug.trace

Syntax:

Debug.trace(String)

Parameters:

String – any value, string or expression. A string should be surrounded by quote marks. This parameter is required.

Remarks:

This function will write a string to the Script Console. This function can be used to help debug an OpusScript program; for example, you can display the value of a variableto check if it is what you expect it to be. The Script Console is not available when a publication is published, therefore this function will have no adverse effect in your program. However, we suggest you comment out any debug statements you enter before you publish the publication.

Example 1:

```
Debug.trace("Hello World!" + "\n")
```

Example 2:

```
Debug.trace(score) // will display the value of the variable score.
```

Example 3:

```
for (var ii = 0; ii < 10; ii++) {
  Debug.trace("ii = " + ii + "\n") }
```

This example will display

```
ii = 0 ii = 1 ii = 2 ii = 3 ii = 4 ii = 5 ii = 6 ii = 7 ii = 8 ii = 9
```

Debug.tracePane

Syntax:

Debug.tracePane(PaneName, String)

Parameters:

PaneName – the name of the pane in the Script Console that the String should be written to. PaneName can be ‘Default’ or ‘Errors’ or any new name you wish to add to the Script Console. PaneName should be surrounded by quote marks. This parameter is required.

String – any value, string or expression. A string should be surrounded by quote marks. This parameter is required.

Remarks:

This function will write a string to a specified pane in the Script Console. This function can be used to help debug an OpusScript program, for example, you can display the value of a variable to check if it is what you expect it to be and place it in a different pane than the ‘Default’ pane. The Script Console is not available when a publication is published, therefore this function will have no adverse effect in your program. However, we suggest you comment out any debug

statements you enter before you publish the publication.

Example:

```
Debug.tracePane("Score Results", score) // will display the value of the
variable score // in a new 'Score Results' pane.
```

IsPreview

Syntax:

IsPreview()

Return:

true if the current player is the Opus Editor preview and false if it is any external player type.

Remarks:

This function tests if the current player is the Opus Editor in Preview mode. It is useful for performing actions while developing a publication (for example; showing variable values in a new window) that you do not want to happen when the publication is published.

Example:

To show an initially hidden text object called textDebug only in the editor preview

```
if (IsPreview()) {
      textDebug.Show(); }
```

WinHelp

Syntax:

WinHelp(Filename, ID)

Parameters:

Filename – Filename of the help file to use. This parameter is required.

ID – Topic ID to jump to. If this parameter is omitted the help contents page is shown.

Remarks:

This function will open a standard Windows help (.hlp) file and display a given topic or the contents page.

Example:

Example 1:

To display the contents tab for a WinHelp file:

```
WinHelp( SYSTEM_PUBLICATION_DIR + "\\MyHelp.hlp" )
```

Example 2:

To display a specific topic in a WinHelp file:

```
WinHelp( SYSTEM_PUBLICATION_DIR + "\\MyHelp.hlp", 254 )
```

HtmlHelp

Syntax:

HtmlHelp(Filename, ID)

Parameters:

Filename – Filename of the help file to use. This parameter is required.

ID – Topic ID to jump to. If this parameter is omitted the help contents page is shown.

Remarks:

This function will open a Compiled HTML Help (.chm) file and display a given topic or the contents page.

Note: CHM support requires Internet Explorer 4 or better. All versions of Windows from Millennium upward (including Windows 2000, XP and Server 2003) support this by default.

Example 1:

To display the contents page for a Compiled HTML Help file:

```
HtmlHelp( SYSTEM_PUBLICATION_DIR + "\\MyHelp.chm" )
```

Example 2:

To display a specific topic in a Compiled HTML Help file:

```
HtmlHelp( SYSTEM_PUBLICATION_DIR + "\\MyHelp.chm", 254 )
```

LaunchSearch

Syntax:

LaunchSearch(PosX, PosY)

Parameters:

PosX – the horizontal coordintate of the window. PosX must be an integer. This parameter is optional.

PosY – the vertical coordinate of the window. PosY must be an integer. This parameter is optional.

Remarks:

This function will open the publication search dialog box on screen.

Note: The Use publication search database option in the General tab of the Publication Properties dialog must be ticked (i.e. turned on) in order for the LaunchSearch function to work when your publication is published.

Example 1:

To open the publication search dialog at a location specifed by the operating system:

```
LaunchSearch()
```

Example 1:

To open the publication search dialog at 100 pixels across and 50 pixels down the screen:

```
LaunchSearch( 100, 50 )
```

LaunchFile

Syntax:

LaunchFile(Filename, Parameters)

Return:

The specified file.

Parameters:

Filename – the pathname of the file to be launched. Filename should be surrounded by quote marks or be a variablecontaining a pathname. This parameter is required.

Note: Pathnames normally contain backslashes e.g. d:\pricelist.doc, all single backslashes should be entered as double backslashes i.e. d:\\pricelist.doc.

Parameters – any command line parameters required by the file that you want to open.

Remarks:

This function allows you to start running a separate program or another Opus publication. You can add optional command line parameters that will be run when the specified file is launched.

Example 1:

To open a MS Word document named pricelist.doc from the CD-ROM drive of the users

machine, se the following syntax:

```
LaunchFile("d:\\pricelist.doc")
```

Example 2:

In Example 1 above, the drive letter was assumed to be d: because that is most often the drive letter assigned to the CD-ROM drive. However, Opus provides a System variable named SYSTEM_CD_DRIVE that will determine the correct drive letter for the user's first CD-ROM. Below is the same example as above but using the System variable

```
var pathname = SYSTEM_CD_DRIVE + "\\pricelist.doc" LaunchFile(pathname)
```

Example 3:

In the example below, the Windows Notepad program is opened showing the Readme.txt file stored in the folder of the current publication's directory. Notice the use of two System variables that automatically work out both the directory where windows is stored on the users machine and the path to the current publication from which this script was run

```
var Fname = SYSTEM_WIN_DIR + "\\notepad.exe" var Params =
SYSTEM_PUBLICATION_DIR + "\\Readme.txt" LaunchFile(Fname,Params)
```

LaunchURL

Syntax:

LaunchURL(URL)

Return:

The specified URL.

Parameters:

URL – the name and path of the URL (Uniform Resource Locator) to be launched. URL should be surrounded by quote marks. This parameter is required.

Remarks:

This function will launch a web browser containing the specified URL.

Example:

```
LaunchURL("www.digitalworkshop.com")
```

SendEmail

Syntax:

SendEmail(ToList, CC, BCC, Subject, Message, Attachment, Showmail, MAPI)

Parameters:

ToList – the email name of the recipient. ToList should be surrounded by quote marks. This parameter is required.

CC – the email address of other recipients. CC should be surrounded by quote marks. This parameter is required but can have a blank entry i.e. just enter the quote marks "".

BCC – the email address of other recipients, the address of these recipients will not be visible to the recepient named in ToList. This parameter is required but can have a blank entry i.e. just enter the quote marks "".

Subject – enter the subject heading for the email. Subject should be surrounded by quote marks. This parameter is required but can have a blank entry i.e. just enter the quote marks "".

Message – the message you want to send in the email. Message can be a string or a variablename containing a string. This parameter is required.

Attachment – the pathname of the file you want to attach to this email. Attachment should be surrounded by quote marks. This parameter is required but can have a blank entry i.e. just enter the quote marks "".

Note: Pathnames normally contain backslashes e.g. d:\\pricelist.doc, all single backslashes should be entered as double backslashes i.e. d:\\pricelist.doc.

Showmail – show the mail dialog box before sending the mail. Showmail can be true or false. If true, the dialog is displayed. If false, the dialog is not displayed and the email is sent immediately. This parameter is optional and the default is true.

MAPI – show the logon dialog for the email system prior to sending the email. MAPI can be true

or false. If true, the MAPI prompt dialog is displayed. If false, the MAPI dialog is not displayed and the email is sent immediately. This parameter is optional and the default is false.

Remarks:

This function will send an email to the specified email addresses, other parameters can be set.

Example:

For a Text object named message that contains the body content of the email message, use the following syntax:

```
message.SetSelection(0,-1) var emailText = message.GetSelectionText()
SendEmail("Jdoe@digitalworkshop.com","","","Welcome" ,emailText)
```

fork

Syntax:

fork(Function, ObjectName)

Parameters:

Function – The name of the function to run in the Script Object. This parameter is required.

ObjectName – The name of the object to be used in the function. This parameter is required.

Remarks:

This function is used to create a new thread for the function to run in, so the command after the function you have called will execute at the same time as the command in the function. Normally, when you call a function in a script the line after the function is not run until the function has completed, the fork function allows both the function and the line after the function to execute at the same time. In this way, the fork function is similar to a Simultaneous action in the Programming tab of the Actions dialog.

Example 1:

In this example, both Image1 and Image2 will display at the same time, unlike a standard script where Image1 would be shown first followed by Image2. This is particularly useful if there is a transition on the images because both images can be displayed simultaneously without you having to wait for the transition on the first image to finish before the next image is displayed.

```
function DisplayObject() {
      this.Show() }
function ShowBothImages() {
                        fork(DisplayObject,Image1) fork(DisplayObject,Image2) }
```

Example 2:

In the second example below, the fork is used to show Image1 and hide Image2 at the same time. This shows that the fork function can be used to call different functions at the same time. Also note that the forked functions do not take parameters, so they use a specific the object name rather that the special keyword this:

```
function ShowObject() {
      Image1.Show() }
function HideObject() {
      Image2.Hide() }
function ShowHideObjects() {
                                                fork(ShowObject) fork(HideObject) }
```

ReadRegistryKey

Syntax:

ReadRegistryKey(RegistryPath, Key)

Return:

The data contained within the Key. For example, if the RegistryPath entered was for Illuminatus Opus Pro and the specified registry key was "Version", the return value would be the current version installed on your computer.

If the key does not exist the return string is empty.

Parameters:

RegistryPath – This is the path where the registry file is stored. This parameter is required and must be surrounded by quote marks.

Note: Type regedit in the Run option in the Start programs at the bottom of the Windows desktop to list the registry keys on your system.

Key – Enter the name of the registry key. The Return Value will be the value of this key. This parameter is required and must be surrounded by quote marks.

Remarks:

The ReadRegistryKey function is used to return the value of a particular registry key contained in the computers registry. This function is often used to dynamically detect the presence of a particular application on an end users computer system. For example, you can check what version of a product is currently installed on a users system – based on this information you can then request they update their version or provide links to latest versions of the software. This is particularly useful if your publication relies on external software to work, such as a PDF reader so people can read your PDF documents.

Note: This function will work when you publish as a standalone or web publication.

Example:

To find the current version of Opus Pro installed on your system, use the following syntax:

```
var OpusVer = ReadRegistryKey("HKEY_LOCAL_MACHINE\\SOFTWARE\\Digit al
Workshop\\Opus Pro", "Version")
```

Note:

In the example above, the command should be typed on one line. Note the use of double backslashes (\\) and not single backslashes to denote a new folder. The Return Value for the variable OpusVer will be the value of the Version key in your Registry for Digital Workshop.

WriteRegistryValue

Syntax:

WriteRegistryValue(Key, ValueName, Value)

Return:

true if the value was written. Otherwise, false.

Parameters:

Key – Full registry key name to store value in. Example: "HKEY_CURRENT_USER\\Software\\oft\\Dates". This parameter is required.

ValueName – Name of the value to store. For Example "Month". This parameter is required.

Value – Value to store. For example "January" or 1. This parameter can be a string or an integer number. All other types are stored as a string. This parameter is required.

Remarks:

This function will only work in the full player - not the screensaver, web plugin or Flex.

You can only write to keys in HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER under the Software subkey - i.e. "HKEY_LOCAL_MACHINE\\Software\\<any key>" or "HKEY_CURRENT_USER\\Software\\<any key>" (or any subkeys thereof)

Example 1:

In this example an integer value is written to the registry:

```
var Value = 42; WriteRegistryValue( "HKEY_CURRENT_USER\\Software\\oft\\Test",
"TheAnswer", Value );
```

Example 2:

In the second example below, a string value is written:

```
var UserName = text1.GetText(); WriteRegistryValue(
"HKEY_CURRENT_USER\\Software\\oft\\Test", "UserName", UserName );
```

DeleteRegistryValue

Syntax:

DeleteRegistryValue(Key, ValueName)

Return:

true if the value was deleted. Otherwise, false.

Parameters:

Key – Full registry key name to delete value from. Example: "HKEY_CURRENT_USER\\Software\\oft\\Test". This parameter is required.

ValueName – Name of the value to delete. This parameter is required.

Remarks:

This function will only work in the full player - not the screensaver, web plugin or Flex.

You can only delete from keys in HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER under the Software subkey - i.e. "HKEY_LOCAL_MACHINE\\Software\\<any key>" or "HKEY_CURRENT_USER\\Software\\<any key>" (or any subkeys thereof)

Example:

In this example a value is deleted from the registry:

```
DeleteRegistryValue( "HKEY_CURRENT_USER\\Software\\oft\\Test", "TheAnswer" );
```

GetINIFileData

Syntax:

GetINIFileData(FilePath, Section, Key)

Return:

The data contained within the key.

If the key does not exist the return string is empty.

Parameters:

FilePath – This is the pathname where the INI file is stored on your computer. This parameter is required and must be surrounded by quote marks. Alternatively, you can use a variablecontaining a pathname.

Section – Enter the name of the section within the INI file in which the key is located. This parameter is required and must be surrounded by quote marks.

Key – Enter the name of the key. The Return Value will be the data contained within this key. This parameter is required and must be surrounded by quote marks.

Remarks:

The GetINIFileData function is used to return the value of a particular key within a specified section of an INI file. This function is used to get specific information from INI files either you have created or are available on a users computer.

Note: This function works when you publish as a standalone publication but not as a web or Flex publication.

Example:

In the example below, the command should be typed on one line. Note the use of double backslashes (\\) and not single backslashes to denote a new folder. The Return Value for the variable KeyValue will be the value of the Name key in section 2 of the INI file named 3d.ini.

```
var KeyValue = GetINIFileData("C:\\Program Files\\Illuminatus Opus
Pro\\Transitions\\3d\\3d.ini", "2", "Name")
```

GetINISectionData

Syntax:

GetINISectionData(FilePath, Section)

Return:

The data contained within the section as a string with each line in the section appearing on a new line.

Parameters:

FilePath – This is the pathname where the INI file is stored on your computer. This parameter is required and must be surrounded by quote marks. Alternatively, you can use a variablecontaining a pathname.

Section – Enter the name of the section within the INI file that you want to view. This parameter is required and must be surrounded by quote marks.

Note: Each section name appears in the INI file surrounded by square brackets. Do not include the square brackets in the string.

Remarks:

The GetINISectionData function is used to return the contents of a section within an INI file. This function is used to get specific information from INI files either you have created or are available on a users computer.

Note: This function works when you publish as a standalone publication but not as a web or Flex publication.

Example 1:

In the example below, the command should be typed on one line. Note the use of double backslashes (\\) and not single backslashes to denote a new folder. The Return Value for the variable sectionData will be a string containing the contents of the section entered in the section parameter

```
var sectionData = GetINISectionData("C:\\Program Files\\Illuminatus Opus Pro\\Transitions\\3d\\3d.ini", "2")
```

Example 2:

To check the contents of the section, use the Debug.trace() function. Continuing the example above

```
Debug.trace(sectionData)
```

If you want to read the data in your program use the Global String functions to search for words in the new string.

PlaySystemSound

Syntax:

PlaySystemSound(Type)

Parameters:

Type – The name of the system sound you want to play. Type must be surrounded by quote marks and must be one of the Types described below – see Type options for more information. This parameter is required.

Remarks:

The PlaySystemSound function is used to play the sound file that has been set for specific system sounds, such as when an error message appears in your program. This function is useful because it uses sound files already on their system, furthermore, if a user has installed a desktop theme that has changed the normal system sounds to sounds of their own choosing, your publication will use the new sounds.

Type options:

The following system sounds should be placed in Type, these should be surrounded by quote marks and only one parameter is allowed per function:

```
PlaySystemSound("Default") PlaySystemSound("Asterisk")
PlaySystemSound("Question") PlaySystemSound("Exclamation")
PlaySystemSound("Error")
```

RGB

Syntax:

RGB(Red, Green, Blue)

Return:

A number indicating a RGB colour value. Each number entered in the red, green and blue parameters are combined to produce a unique number (i.e. RGB colour value) that will generate a different colour.

Parameters:

Red – A number indicating the amount of red used in the colour. The number can be any number in the range 0 to 255. 0 is equal to no amount of red used to make the colour and 255 is a full red value. The number must be an integer. This parameter will also accept a variable containing an integer. This parameter is optional and the default value is 0.

Green – A number indicating the amount of green used in the colour. The number can be any number in the range 0 to 255. 0 is equal to no amount of green used to make the colour and 255 is a full green value. The number must be an integer. This parameter will also accept a variable containing an integer. This parameter is optional and the default value is 0.

Blue – A number indicating the amount of blue used in the colour. The number can be any number in the range 0 to 255. 0 is equal to no amount of blue used to make the colour and 255 is a full blue value. The number must be an integer. This parameter will also accept a variable containing an integer. This parameter is required. Default value is 0.

Remarks:

The RGB function allows you to generate a colour value that can then be applied to objects on a page using other OpusScript functions, such as, SetColour, SetBtnColour and SetFlare. You can make any combination of colour by setting the amount of red, green and blue in the parameters. This is particularly useful if you want to change the colour of objects displayed based on a user's preference.

Example:

To set the RGB value to green and then change the colour of a Text object named score to green, use the following syntax:

```
ScoreCol = RGB(0,255,0) UserScore.SetColour(ScoreCol)
```

Note: A simply way of getting the RGB value for a colour is to move the mouse over the colour in the Colour Palette at the bottom of the Opus Editor – this will display the RGB values in a tooltip.

GetJoystickState

Syntax:

GetJoystickState()

Return:

A new object with the following properties:

x - the x-axis. The range is from -1.0 to +1.0.

y - the y-axis The range is from -1.0 to +1.0.

rz - the z-axis (normally twist) The range is from -1.0 to +1.0.

s - the slider (normally throttle control) The range is from -1.0 to +1.0.

pov - the POV (Point of View) 'hat' position. This is given as an angle in the range 0-360 degrees, plus a special value of –1, which represents centred.

f1 - Fire button 1. 1 is pressed, 0 is not pressed.

f2 - Fire button 2. 1 is pressed, 0 is not pressed.

f3 - Fire button 3. 1 is pressed, 0 is not pressed.

f4 - Fire button 4. 1 is pressed, 0 is not pressed.

Remarks:

This function gets the current state of the first joystick available.

Example:

```
var State = GetJoystickState() if (State.f1) {
      // Do something... }
```

CountLines

Syntax:

CountLines(Name)

Return:

The number of lines in the specified Text object or variable

Parameters:

Name – set the name of object in which you want to count the lines. Name can either be the name of a Text object or of a variablename. The name must NOT be surrounded by quote marks. This parameter is required.

Remarks:

This function will return the number of lines in the specified Text object or named variable.

Example 1:

For an object (or variable) named myText, use the following syntax:

```
var numOfLines = CountLines(myText)
```

Example 2:

```
var numOfLines = CountLines("Line1\nLine2\nLine3")
```

CountWords

Syntax:

CountWords(Name)

Return:

The number of words in the specified Text object or variable

Parameters:

Name – set the name of object in which you want to count the number of words. Name can either be the name of a Text object or of a variable name. The name must NOT be surrounded by quote marks. This parameter is required.

Remarks:

This function will return the number of words in the specified Text object or named variable.

Example 1:

For an object (or variable) named myText, use the following syntax:

```
var numOfWords = CountWords(myText)
```

Example 2:

```
var numOfWords = CountWords("Test string – could also be a variable")
```

Global String - Overview

The Global String functions allow simple string manipulation operations in a script. These functions are all properties of the Global String object, which means that in the script they must be called as follows:

String.functionName

This means, type the word String followed by the name of one of the OpusScript functions listed below separated by a full stop, For example:

```
var name = "Mr Smith" var title = String.left(name, 2)
```

Note: All of the Global String functions can be used directly in a script with other functions such as, variable, loop conditions, and so on.

Function names are case-sensitive and must be typed as they appear in the Functions heading below.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Global String objects:

Class Hierarchy:

Global String

Functions:

bool Convert a value to a Boolean

contains Test if one string contains another string

format Format a number as a string

integer Convert a value to an integer

left Get the left end of a string

length Get the length of a string

mid Get part of the middle of a string

number Convert a value to a number

random Get a random integer

right Get the right end of a string

string Convert a value to a string

tolower Get a lower case version of a string

toupper Get an upper case version of a string

word Get a word from a string

left

Syntax:

left(String, Chars)

Return:

A string consisting of the given number of characters from the left end of the string.

Parameters:

String - The String can be a string of text surrounded by quote marks or a variablename containing a string.

Chars - The number of characters from the left of the String to return in the Return Value. Chars can be a number or a variable name containing a number.

Remarks:

If the number of characters specified is more than the length of the string the whole string is returned. Both parameters are required.

Example:

```
var name = "Mr Smith" var title = String.left(name,2)
```

Note: The Return Value in the example above would be "Mr". Therefore, the variable title will contain the value "Mr".

right

Syntax:

right(String, Chars)

Return:

A string consisting of the given number of characters from the right end of the string.

Parameters:

String – The String can be a string or a variablename containing a string.

Chars - The number of characters from the right of the String to return in the Return Value. Chars can be a number or a variable name containing a number.

Remarks:

If the number of characters specified is more than the length of the string the whole string is returned. Both parameters are required.

Example:

```
var name = String.right("Mr Smith",5)
```

Note: The Return Value in the example above would be "Smith". Therefore, the variable name

will contain the value "Smith".

mid

Syntax:

mid(String, Index, Chars)

Return:

A string consisting of the given number of characters starting from the Index position in the string.

Parameters:

String - The String can be a string of text surrounded by quote marks or a variablename containing a string. This parameter is required.

Index - The Index position of the first character to return i.e. the first character is Index position 1, the second character Index position 2, and so on. Index can be a number or a variable name containing a number. This parameter is required.

Chars - The number of characters from the left of the String starting from the Index position. This parameter is optional, If this parameter is omitted all the characters to the end of the string are returned. Chars can be a number or a variable name containing a number.

Remarks:

If the number specified in Chars would exceed the length of the String, the whole of the string from the Index position is returned.

Example:

```
var name = "Mr James Smith" var firstName = String.mid(name,4,5)
```

Note: The Return Value in the example above would be "James". Therefore, the variable firstName will contain the value "James".

toupper

Syntax:

toupper(String)

Return:

The complete String with all lower case letters converted to upper case.

Parameters:

String - The string to convert. The String can be a string of text surrounded by quote marks or a variable name containing a string. This parameter is required.

Example:

```
var surname = String.toupper("Smith") var firstName =
```

```
String.toupper(String.mid("Mr James Smith",4,5))
```

Note: The Return Value in the first example above would be "SMITH". Therefore, the variable surname will contain the value "SMITH". In the second example, two string functions have been combined, the Return Value for the toupper function will be "JAMES". Therefore, the variable firstName will contain the value "JAMES".

tolower

Syntax:

tolower(String)

Return:

The complete String with all characters converted to lower case.

Parameters:

String - The string to convert. The String can be a string of text surrounded by quote marks or a variable name containing a string. This parameter is required.

Example 1:

```
var surname = String.tolower("Smith")
```

Example 2:

```
var firstName = String.tolower(String.mid("Mr James Smith",4,5))
```

Note: In the first example above the variable surname will contain the value "smith". In the second example, two string functions have been combined, the variable firstName will contain the value "james".

length

Syntax:

length(String)

Return:

The length of the String.

Parameters:

String - The String can be a string of text surrounded by quote marks or a variable name containing a string. This parameter is required.

Remarks:

This function can be useful when determining the number of times a loop should run.

Example:

```
var name = "Smith" var numLoops = String.length(name) for (I = 0; I <=
numLoops; I++) {
```

```
    Debug.trace(I + "\n") }
```

Note: In this example, the Return Value of the length function is 5, note the loop will repeat six times because the first loop starts at 0.

contains

Syntax:

contains(String, Term, IgnoreCase)

Return:

true if the value in Term is found in String. Otherwise false.

Parameters:

String - The string to be searched. The String can be any string surrounded by quote marks or a variable name containing a string.

Term - The string to search for within String. The Term can be any string surrounded by quote marks or a variable name containing a string.

IgnoreCase – If true the comparison will ignore the case of the characters being compared; if false then the comprison is case sensitive. This parameter is optional and the default value is false.

Remarks:

Both parameters are required for this function. This function is case- sensitive, in order to return the value true, the value of Term must not only match part (or all) of the value of String, it must also be in the same upper or lower case letters.

Example 1:

A case sensitive comparison:

```
var fullName = "Ms Jane Doe" var firstName = String.contains(name, "Jane")
```

Note: In this example, firstname is true because the term ‘Jane’ is contained within the variable name. If the term ‘jane’ had been used, firstname would be false.

Example 2:

A case insensitive comparison:

```
if ( String.contains("This is a Test", "test"), true ) {
  Debug.trace("Found"); } else {
  Debug.trace("Not found"); }
```

Note: In this example, “Found” is displayed in the script output window because the IgnoreCase flag is set to true and the term ‘test’ is contained within the string. If the IgnoreCase flag was set to false, “Not found” would be displayed.

word

Syntax:

word(String, Index, Separator)

Return:

A single word from String. If the word specified by Index does not exist, and empty string is returned.

Parameters:

String - The string to be searched. The String can be any string surrounded by quote marks or any variable name containing a string. This parameter is required.

Index - The Index position of the word to return. The first word in String is Index position 1, the second word is Index position 2, and so on. Index must be entered as an integer. This parameter is required.

Separator - A string consisting of characters that separate words. This parameter is optional. If a Separator is not specified then words within String are assumed to be separated by spaces.

Example 1:

```
var fullName = "Mr James Henry Smith" var middleName =
String.word(fullName,3)
```

In this example, middleName will contain the value "Henry"

Example 2:

```
var webAddress = "www.digitalworkshop.com" var compName =
String.word(webAddress,2,".")
```

In this example, the optional Separator parameter is used, compName will contain "digitalworkshop".

format

Syntax:

format(Format, Number)

Return:

The number expressed as a string in the given format.

Parameters:

Format - A string describing the format to express the number in. The string should be of the form "Digits.Decimals", where Digits is the total number of digits to use for the number (including the decimal point if there is one) and Decimals is the number of decimal places to use. If the Format string starts with 0 (zero) then the resulting field will be padded with leading zeroes to fill the requested width if it does not already do so.

Number – The number you want to format.

Example 1:

```
var pi_str = String.format("4.2",Math.PI) // would set pi_str to the string "3.14"
```

Example 2:

```
var pi_str = String.format("05.2",Math.PI) // would set pi_str to the string "03.14"
```

bool

Syntax:

bool(Value)

Return:

The parameter converted to a Boolean (true or false).

Parameters:

Value - The value is any valid expression, string or variable name. This parameter is required.

Remarks:

Usually used to convert strings to a Boolean.

Example:

```
var answer = 3.14 var correct = String.bool(3.14 == answer) if (correct) {
  feedback.ReplaceSelection("Your answer was correct!") } else {
  feedback.ReplaceSelection("Your answer was incorrect.") }
```

In this example, correct is true, because the variable answer equals 3.14. The if statement will evaluate to true and replace the text in the Text object named feedback with ‘Your answer was correct!"

integer

Syntax:

integer(Value)

Return:

The parameter converted to an integer.

Parameters:

Value - The Value is any valid expression, string or variable name. This parameter is required.

Remarks:

Usually used to convert strings to an integer.

```
Example 1: var answer = String.integer("34")
```

In this example, answer will contain the number 34

Example 2:

```
var num1 = 3.1415926 var num2 = 2.4567 var totalInt = String.integer(num1 * num2)
```

In this example, totalInt will contain the number 7

number

Syntax:

number(Value)

Return:

The parameter converted to a number.

Parameters:

Value - The Value is any valid expression, string or variable name. This parameter is required.

Remarks:

Usually used to convert strings to a number.

Example:

```
var answer = String.number("5.4")
```

In this example, answer is 5.4.

string

Syntax:

string(Value)

Return:

The parameter converted to a string i.e. the number data type is converted to a string data type.

Parameters:

Value - The Value is any valid expression, string or variable name. This parameter is required.

Example:

```
var num1 = String.string(3.1415) var num2 = 926 var total = num1 + num2
```

In this example, total is "3.1415926" because num1 has been converted to a string, therefore, the + symbol in the total expression is considered to be a concatenation operator and not an addition operator.

random

Syntax:

random(Limit)

Return:

A random integer between 0 and Limit - 1 inclusive.

Parameters:

Limit - The Limit must be an integer or a variable name containing an integer value. The random integer returned will range from 0 to Limit - 1. This parameter is required.

Example:

```
var randomNum = String.random(100)
```

In this example, randomNum will be any integer between 0 and 99.

Graphical Objects - Overview

All objects that can visibly appear on a page in a publication can use the Graphical Object functions. This includes:

Text Objects

Buttons

Videos

Slideshows

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Graphical Objects

Basic Objects

Functions:

CaptureMouse Capture the mouse input

CloneObject Clone an object

CopyToClipboard Copy an object to the Windows clipboard

DestroyClonedObject Destroy a cloned object

Enable Enable or disable this object

Fade Fade an object by an amount

FollowPath Make an object follow an animation path

GetAppearance Get the appearance information for the object State of a specified object

GetDisplayData Get display information related to an object e.g. scale and skew

GetHeight Get the height of the object

GetLayer Get the layer that the object is on

GetObjectDimensions Get the position and size of an object

GetPersistentObject Get an object for storing information that will persist when the page is not visible

GetPosition Get the x and y coordinate of the objects position

GetPosition Get the x and y coordinate of an animation path’s length

GetPositionFromPercent Get the x and y coordinate of an animation path from a percent along the path

GetScrollInfo Get the scroll position, scroll size and scrollbar

GetTotalLength Get the total length of an animation path

GetWidth Get the width of the object

GetXPosition Get the x coordinate of the objects position

GetYPosition Get the y coordinate of the objects position

Hide Hide this object

IsEnabled Test if this object is enabled

IsObjectIntersecting Test if this object is intersecting another

IsShowing Test if this object is showing

Move Move the object by its x and y coordinates

MoveX Move the object by its x coordinate

MoveY Move the object by its y coordinate

ReleaseMouse Release the mouse input

RemoveAlpha Delete the Alpha effect on a specified object

RemoveBackground Delete the Background style on a specified object

RemoveBorder Delete the Border style on a specified object

RemoveBtnColour Delete the Button style on a specified Button object

RemoveFlare Delete the Flare effect on a specified object

RemoveImage Delete the Image on a specified object

RemoveShadow Delete the Shadow effect on a specified object

RemoveTexture Delete the Texture effect on a specified object

ResetAnimation Reset an animation

Roll Roll an object by a specified angle

Rotate Rotate an object by a specified angle

Scale Scale an object horizontally and vertically by a percentage

ScaleH Scale an object horizontally by a percentage

ScaleV Scale an object vertically by a percentage

SetAlpha Create an Alpha effect on a specified object

SetBackground Create a Background style on a specified object

SetBorder Create a Border style on a specified object

SetBtnColour Create a Button style on a specified object

SetDisplayData Set display information related to an object e.g.

SetFlare Create a Flare effect on a specified object

SetFocus Set the keyboard input focus

SetImage Insert an Image on a specified object

SetLayer Set the layer that the object is on

SetObjectDimensions Set the position and size of an object

SetPosition Set aspects of the objects position

SetPositionX Set the x coordinate of the objects position

SetPositionY Set the y coordinate of the objects position

SetRoll Set an objects roll to a specified angle

SetRotation Set an objects rotation to a specified angle

SetScale Set the horizontal and vertical scale of an

object to a percentage

SetScaleH Set the horizontal scale of an object to a percentage

SetScaleV Set the vertical scale of an object to a percentage

SetScrollPosition Set the scroll position of the object

SetShadow Create a Shadow effect on a specified object

SetSkew Skew an object horizontally and vertically to a specified angle

SetSkewH Skew an object horizontally to a specified angle

SetSkewV Skew an object vertically to a specified angle

SetSpin Set an objects spin to a specified angle

SetTexture Create a Texture effect on a specified object

SetTransparency Set the objects transparency

Show Show this object

Skew Set the skew of an object horizontally and vertically to a specified angle

SkewH Set the skew of an object horizontally to a specified angle

SkewV Set the skew of an object vertically to a specified angle

Spin Spin an object by a specified angle

StopAnimation Stop an animation

Show

Syntax:

Show(ResetPosition)

Parameters:

ResetPosition – If an object has been moved via other scripting by default showing the object will reset it to its initial position. ResetPosition can be true or false. If true, the object is reset to its original position. If false, the object is not reset to its original position. This parameter is optional and the default is true.

Remarks:

This function will show the specified object on the page. If the specified object shows with a transition effect the next line of code in the Script Action or Script Object will not execute until the object has finished showing.

Example 1:

```
myImage.Show()
```

Example 2:

```
myImage.Move(20,30,0.5) myImage.Hide() wait(2) myImage.Show(false)
```

Hide

Syntax:

Hide()

Remarks:

This function will hide the specified object on the page. No parameters are required. If the specified object hides with a transition effect, the next line of code in the Script Action or Script Object will not execute until the object has finished hiding.

Example:

```
myImage.Hide()
```

IsShowing

Syntax:

IsShowing()

Return:

true if the specified object is currently showing. Otherwise false.

Remarks:

This function will check if the specified object is currently displayed on the page. No parameters are required.

Example:

```
var imageShow = myImage.IsShowing()
```

Note: In this example, the variable imageShow will contain true or false depending on the current state of myImage.

Enable

Syntax:

Enable(Set)

Parameters:

Set – Sets the specified object's Enable status to enabled if Set is true or disabled if Set is false. This parameter is required. Set should either be true or false or a variable that evaluates to true or false.

Remarks:

This function will enable or disable the specified object. To disable the object set the Set parameter to false, to enable the object set it to true.

Example 1:

```
button1.Enable(false)
```

Example 2:

```
var status = false button1.Enable(status)
```

Note: For a Button object named button1, either of the above example will disable the button.

IsEnabled

Syntax:

IsEnabled()

Return:

true if the specified object is enabled. Otherwise false if the object is disabled.

Remarks:

This function checks the current Enable status of the specified Object. No parameters required.

Example:

```
var checkStatus = button1.IsEnabled()
```

Note: In this example, the variable checkStatus will contain true if the button is enabled or false if it is disabled.

IsObjectIntersecting

Syntax:

IsObjectIntersecting(OtherObject)

Return:

true if the specified object intersects OtherObject. Otherwise false.

Parameters:

OtherObject – The name of another object on the current page. This parameter is required. OtherObject can be the name of another object on the current page.

Remarks:

This function tests if one object is intersecting another object on the current page.

Example:

```
var test = text1.IsObjectIntersecting(text2)
```

For two Text objects named text1 and text2 use the following syntax:

Note: In the example above, test will either be true or false.

GetPersistentObject

Syntax:

GetPersistentObject()

Return:

A unique persistent copy of this object.

Remarks:

The first time this function is called for any particular object a new script object is created as if the script "new Object" had been used. Subsequent calls to the function will return the same object. By setting properties of the object returned information can be stored that will remain even while the page is not visible. This mechanism is required because the objects on the page are destroyed when the page is no longer visible and are recreated when it becomes visible again.

Example:

```
var Persist = text1.GetPersistentObject()
```

GetObjectDimensions

Syntax:

GetObjectDimensions()

Return:

A new object with the following properties:

top - the y coordinate of the top edge of the specified object.

bottom - the y coordinate of the bottom edge of the specified object.

left - the x coordinate of the left edge of the specified object.

right - the x coordinate of the right edge of the specified object.

width - the width of the specified object

height – the height of the specified object.

Remarks:

This function returns the position and size of a specified object in a new Object.

Example 1:

For an Image object named Image1, use the following syntax to create a new object named dimInfo that will contain the Return Values for the properties listed

```
var dimInfo = Image1.GetObjectDimensions()
```

Example 2:

In this example, the Return Values of the properties for image1 become properties of the new object dimInfo. To reference a particular property within the new Object, use the following syntax:

```
var ImageTop = dimInfo.top var ImageBottom = dimInfo.bottom
```

SetObjectDimensions

Syntax:

SetObjectDimensions(DimensionObject)

Parameters:

DimensionObject – DimensionObject is the name of the new object created using the GetObjectDimensions function. The properties of DimensionsObject will replace the properties of the specified object in this SetObjectDimensions. This parameter is required.

Remarks:

This function sets the dimensions of a specified object to the same dimensions as another Object. The dimensions it sets are: left, right, top, bottom, width and height. This function requires one parameter which is the name of the new object created by a GetObjectDimensions function.

Example:

To set the dimensions of an Image object named Image1 to Image2, use the following syntax:

```
var dimInfo = Image1.GetObjectDimensions()
Image2.SetObjectDimensions(dimInfo)
```

SetObjectSize

Syntax:

SetObjectSize(Width, Height)

Parameters:

Width – set the width of the.object in pixels Width is an integer.

Height – set the height of the.object in pixels Height is an integer.

Remarks:

This function changes the original width and height of an object. This is different to scaling an object via the SetScale() function.

Example:

To set the size of an Image object named Image1 to use the following syntax:

```
Image1.SetObjectSize( 20, 20 )
```

GetPosition

Syntax:

GetPosition()

Return:

A new object containing the x and y coordinates of the specified Object, with respect to the top left corner of the page.

Remarks:

This function returns the x and y coordinates of a specified object in a new Object. No Parameters required.

Example:

For a Button object named myButton, use the following syntax:

```
var pos = myButton.GetPosition() var PosX = pos.x var PosY = pos.y
```

GetXPosition

Syntax:

GetXPosition()

Return:

The x coordinate of the centre of the specified object with respect to the top left corner of the page.

Remarks:

This function returns the x coordinate of a specified Object. No parameters are required. This function is normally used with the GetYPosition function to show the movement of Objects around a page.

Example:

For a Frame named frame1, use the following syntax to show the x coordinate in a page variable named x_coord

```
x_coord = frame1.GetXPosition()
```

Note: In this example, the page variable x_coord should already have been created on your page and it will display the x coordinate (i.e. the Return Value) for Object frame1.

GetYPosition

Syntax:

GetYPosition()

Return:

The y coordinate of the centre of the specified object with respect to the top left corner of the page.

Remarks:

This function returns the y coordinate of a specified Object. No parameters are required. This function is normally used with the GetXPosition function to show the movement of Objects around a page.

Example:

For a Frame named frame1, use the following syntax to show the y coordinate in a page variable named y_coord

```
y_coord = frame1.GetYPosition()
```

GetWidth

Syntax:

GetWidth()

Return:

The width of the specified Object.

Remarks:

This function returns the width of the specified object in pixels. No parameters are required. This function is normally used with the GetHeight function.

Example:

```
var objWidth = button1.GetWidth()
```

GetHeight

Syntax:

GetHeight()

Return:

The height of the specfied Object.

Remarks:

This function returns the height of the specified object in pixels. No parameters are required. This function is normally used with the GetWidth function.

Example:

For a Button object named button1, use the following syntax:

```
var objHeight = button1.GetHeight()
```

GetDisplayData

Syntax:

GetDisplayData()

Return:

A new object with the following properties:

x - the x coordinate of the centre of the specified object, as an offset from the original x position.

y - the y coordinate of the centre of the specified object, as an offset from the original y position.

scalex - the amount the specified object has been scaled in the x direction from its original size.

scaley - the amount the specified object has been scaled in the y direction from its original size.

angle - the angle in degrees the specified object has been rotated to.

anglex - the angle in degrees the specified object has been rotated in 3D about the x axis.

angley - the angle in degrees the specified object has been rotated in 3D about the y axis.

skewx - the amount the specified object has been skewed in the x direction.

skewy - the amount the specified object has been skewed in the y direction.

transparency - the specified object's percentage transparency i.e. 100 is transparent and 0 is opaque.

Remarks:

This function returns all the display information about the specified object in a new Object.

Example:

For an Image object named Image1, use the following syntax to create a new object named positionInfo that will contain the Return Values for the properties listed

```
var positionInfo = Image1.GetDisplayData()
```

Note: In this example, the Return Values of the properties for image1 become properties of the new object positionInfo. To reference a particular property within the new Object, use the following syntax:

```
var ImageAngle = positionInfo.angle var ImageTrans =
positionInfo.transparency
```

SetDisplayData

Syntax:

SetDisplayData(PositionObject)

Parameters:

PositionObject – PositionObject is the name of the new object created using the GetDisplayData function. The properties of PositionObject will replace the properties of the specified object in

this SetDisplayData function. Any previous new object created with a GetDisplayData function can be used as a PositionObject. This parameter is used instead of PosX and PosY.

Remarks:

This function is used to either set the x and y coordinates of the specified object or to set all aspects of the specified Objects position. The properties you can set are the same as those listed in the GetDisplayData function.

Example 1:

In this example, Image2 is set to move 200 pixels across and 150 pixels down. For an Image object named Image2 use the following syntax:

```
var DisplayObj = new Object() DisplayObj.x = 200 DisplayObj.y = 150
Image2.SetDisplayData( DisplayObj )
```

Example 2:

In the next example, Image2 is going to be set the same properties as Image1 shown in the GetDisplayData function; the new object created for Image1 is named positionInfo. Use the following syntax to set the Image1 properties to Image2

```
var PositionInfo = Image2.GetDisplayData() Image2.SetDisplayData(
PositionInfo )
```

GetAppearance

Syntax:

GetAppearance(Appearance)

Return:

A new object that contains specific appearance properties for the specified object's Object State. In other words, if for example you want to change the flare of an object when you mouse over it, you must create a new object using the GetAppearance function, specifying Mouse Over in the Appearance parameter. The new object created by this function will include a Flare property, which you can use with the SetFlare function to create a flare or the RemoveFlare function to delete a flare already set for the specified object.

The new object contains properties that allow you to set or remove the following appearances: Alpha, Background, Border, Button Colour, Flare, Image, Shadow and Texture.

The new object is used with the following OpusScript functions:

To create a new appearance for a specified object: SetAlpha, SetBackground, SetBorder, SetBtnColour, SetFlare, SetImage, SetShadow and SetTexture.

To remove an appearance for a specified object: RemoveAlpha, RemoveBackground, RemoveBorder, RemoveBtnColour, RemoveFlare, RemoveImage, RemoveShadow and RemoveTexture.

Parameters:

Appearance – the name of the Object State you want to return. Appearance must be a string or a variable containing one of the following key words: Default, Mouse Over, Mouse Pressed or Disabled. This parameter is optional and the default is Default if no other key word is used.

Remarks:

This function will return a new object containing the current appearance settings for a specified object's Object State. The Appearance parameter is used to specify the Object State you want to return (e.g. Mouse Over), the new object created by this function will then contain the current appearance settings you have set for the specified object when the user moves the mouse over it. This function is particularly useful because it allows you to change the appearance of an object while the user is running the publication.

Note: This function is equivalent to the QuickBuild menu options. A full description of the QuickBuild menu is provided in the main Opus Help file. To open the help, click on Contents & Index option in the Help menu at the top of the Opus Editor. Search for the word QuickBuild in the Index tab of the help file for more information.

Example 1:

To change the appearance of an object when the user moves the mouse over it, you must first create a new object using the GetAppearance function. For a Button object named option1, use the following syntax:

```
ButAppearanceMO = option1.GetAppearance("Mouse Over")
```

Note: The variable ButAppearanceMO will now contain the mouse over appearance properties: Alpha, Background, Border, Button Colour, Flare, Image, Shadow, Texture and Transparency.

Example 2:

Using the new object created in Example 1 above, we can now set new appearance for properties or remove existing properties. In the expanded example below we have created the new object and removed the flare for the Button object named option1 and changed the button's surface colour.

```
ButAppearanceMO = option1.GetAppearance("Mouse Over")
ButAppearanceMO.RemoveFlare() ButCol = RGB(0,0,255)
ButAppearanceMO.SetBtnColour(ButCol)
```

Note: The SetBtnColour function requires a colour value for the surface colour – in this example, we have used the RGB function to set the colour to blue. The SetBtnColour function also allows you to change other parameters as well, such as its light bevel colour and bevel width but these are optional parameters and ignored for the purpose of this example.

RemoveAlpha

Syntax:

RemoveAlpha()

Remarks:

This function will remove the Alpha effect currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Alpha effect will be

removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Alpha effect is removed for the Object State of the object specified in the GetAppearance function.

Note: The Alpha effect can be set using the SetAlpha function or in the Effects tab of the object's Properties dialog.

Example 1:

To remove the Alpha effect on the Normal Object State for an Image object named IntroGraph, use the following syntax:

```
IntroGraph.RemoveAlpha()
```

Example 2:

To remove the Alpha effect on the Mouse Over Object State for an Image object named IntroGraph, use the following syntax:

```
DelAlpha = IntroGraph.GetAppearance("Mouse Over") DelAlpha.RemoveAlpha()
```

Note: When the user moves the mouse over the object named IntroGraph, the Alpha effect will be the same as the object's Normal Object State.

RemoveBackground

Syntax:

RemoveBackground()

Remarks:

This function will remove the Background colour currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Background colour

will be removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Background colour is removed for the Object State of the object specified in the GetAppearance function.

Note: The Background colour can be set using the SetBackground function or in the Background tab of the object's Properties dialog.

Example 1:

To remove the Background colour on the Normal Object State for a Text object named Intro, use the following syntax:

```
Intro.RemoveBackground()
```

Example 2:

To remove the Background colour on the Mouse Over Object State for a Text object named Intro, use the following syntax:

```
DelBckgrnd = Intro.GetAppearance("Mouse Over") DelBckgrnd.RemoveBackground()
```

Note: When the user moves the mouse over the object named Intro, the Background colour will be the same as the object's Normal Object State.

RemoveBorder

Syntax:

RemoveBorder()

Remarks:

This function will remove the Border style currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Border style will

be removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Border style is removed for the Object State of the object specified in the GetAppearance function. Note:

The Border style can be set using the SetBorder function or in the Border tab of the object's Properties dialog.

Example 1:

To remove the Border style on the Normal Object State for a Text object named Intro, use the following syntax:

```
Intro.RemoveBorder()
```

Example 2:

To remove the Border style on the Mouse Over Object State for a Text object named Intro, use the following syntax:

```
DelBorder = Intro.GetAppearance("Mouse Over") DelBorder.RemoveBorder() Note:
```

When the user moves the mouse over the object named Intro, the Border style will be the same as the object's Normal Object State.

RemoveBtnColour

Syntax:

RemoveBtnColour()

Remarks:

This function will remove the Button style currently applied to the specified object. The Button style's removed will be the buttons: surface colour; Light bevel colour; Dark bevel colour; Bevel width; Bevel transparency; and Border.

There are two methods of using this function:

(i) Specify an object name – in this method, the Button style will

be removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Button style is removed for the Object State of the object specified in the GetAppearance function. Note:

The Button style can be set using the SetBtnColour function or in the Button tab of the object's Properties dialog.

Example 1:

To remove the Button style on the Normal Object State for a Button object named Option1, use the following syntax:

```
Option1.RemoveBorder()
```

Example 2:

To remove the Button style on the Mouse Over Object State for a Button object named Option1, use the following syntax:

```
DelButton = Option1.GetAppearance("Mouse Over") DelButton.RemoveBorder()
Note:
```

When the user moves the mouse over the object named Option1, the Button style will be the same as the object's Normal Object State.

RemoveFlare

Syntax:

RemoveFlare()

Remarks:

This function will remove the Flare style currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Flare style will

be removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Flare style is removed for the Object State of the object specified in the GetAppearance function. Note:

The Flare style can be set using the SetFlare function or in the Effects tab of the object's Properties dialog.

Example 1:

To remove the Flare style on the Normal Object State for a Button object named Option1, use the following syntax:

```
Option1.RemoveFlare()
```

Example 2:

To remove the Flare style on the Mouse Over Object State for a Button object named Option1, use the following syntax:

```
DelButton = Option1.GetAppearance("Mouse Over") DelButton.RemoveFlare() Note:
```

When the user moves the mouse over the object named Option1, the Flare style will be the

same as the object’s Normal Object State.

RemoveImage

Syntax:

RemoveImage()

Remarks:

This function will remove the Image file currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Image file will be

removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Image file is removed for the Object State of the object specified in the GetAppearance function. Note:

The Image file can be set using the SetImage function or in the Image tab of the object's Properties dialog.

Example 1:

To remove the Image file on the Normal Object State for an Image object named IntroGraph, use the following syntax:

```
IntroGraph.RemoveImage()
```

Example 2:

To remove the Image file on the Mouse Over Object State for an Image object named IntroGraph, use the following syntax:

```
DelGraphic = IntroGraph.GetAppearance("Mouse Over") DelGraphic.RemoveImage()
Note:
```

When the user moves the mouse over the object named IntroGraph, the Image file used will be the same as the object's Normal Object State.

RemoveShadow

Syntax:

RemoveShadow()

Remarks:

This function will remove the Shadow style currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Shadow style will be

removed from the Normal (i.e. default) Object State

(ii) Specify a new object created with the GetAppearance function – in this method, the Shadow style is removed for the Object State of the object specified in the GetAppearance function. Note:

The Shadow style can be set using the SetShadow function or in the Effects tab of the object's Properties dialog.

Example 1:

To remove the Shadow style on the Normal Object State for a Text object named Intro, use the following syntax:

```
Intro.RemoveShadow()
```

Example 2:

To remove the Shadow style on the Mouse Over Object State for a Text object named Intro, use the following syntax:

```
DelShadow = Intro.GetAppearance("Mouse Over") DelShadow.RemoveShadow() Note:
```

When the user moves the mouse over the object named Intro, the Shadow style used will be the same as the object's Normal Object State.

RemoveTexture

Syntax:

RemoveTexture()

Remarks:

This function will remove the Texture effect currently applied to the specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Texture effect

will be removed from the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Texture effect is removed for the Object State of the object specified in the GetAppearance function. Note:

The Texture effect can be set using the SetTexture function or in the Effects tab of the object's Properties dialog.

Example:

Example 1:

To remove the Texture effect on the Normal Object State for an Image object named IntroGraph, use the following syntax:

```
IntroGraph.RemoveTexture()
```

Example 2:

To remove the Texture effect on the Mouse Over Object State for an Image object named IntroGraph, use the following syntax:

```
DelTexture = IntroGraph.GetAppearance("Mouse Over")
DelTexture.RemoveTexture() Note:
```

When the user moves the mouse over the object named IntroGraph, the Texture effect used will be the same as the object's Normal Object State.

SetAlpha

Syntax:

SetAlpha(Style, Transparency1, Transparency2)

Parameters:

Style – the name of one of the Alpha styles you can create with this function. Style must be one of the following keywords: None, Horizontal, Vertical, NWSE, NESW, Centre or Circle. This parameter is required. The keyword must be surrounded by quote marks or a variable containing one of the valid keywords.

Transparency1 – a percentage, showing the minimum transparency you want to set on the alpha channel. Transparency1 must be an integer between 0 and 100. This parameter is optional and the default is 0.

Transparency2 – a percentage, showing the maximum transparency you want to set on the alpha channel.

Transparency2 must be an integer between 0 and 100. This parameter is optional and the default is 100.

Remarks:

This function allows you to create the Alpha effect for a specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Alpha effect will

be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Alpha effect is set for the Object State of the object specified in the GetAppearance function. Note:

The Alpha effect can be removed using the RemoveAlpha function or in the Effects tab of the object's Properties dialog.

Example:

Example 1:

To set the Alpha effect on the Normal Object State for an Image object named IntroGraph to glass with a maximum transparency of 100% use the following syntax:

```
IntroGraph.SetAlpha("Horizontal",0,100)
```

Example 2:

To set the Alpha effect on the Mouse Over Object State for an Image object named IntroGraph, use the following syntax:

```
createEffect = IntroGraph.GetAppearance("Mouse Over")
createEffect.SetAlpha("Glass",0,100)
```

SetBackground

Syntax:

SetBackground(ObjectName)

Parameters:

ObjectName – the name of an object that can contain any of the following properties:

Style – one of the Background styles you can create with this function. Style must be one of the following keywords: solid, linear, centre, circular, radial, spiral, spiral2, exponentialspiral or concentric. This property is optional. The keyword must be surrounded by quote marks or a variable containing one of the valid keywords.

Colour – the colour of the background. For a solid background style this is a single valid RGB value or a variable containing a valid RGB value. For any other gradient style (e.g. spiral) this is an array of RGB colour values – see example.

The RGB value can be calculated using the RGB function.

transparency – a percentage, showing the level of transparency for the background. transparency must be an integer between 0 and 100. This property is optional.

For non-solid styles, you can also set the following properties:

angle – the angle in degrees for linear, radial, spiral, spiral2 and exponentialspiral styles. angle must be a number between 0 and 359. This property is optional.

This property does not apply to centre, circular or concentric styles.

twist – the amount of twist to apply to spiral, spiral2 and exponentialspiral gradient styles. twist must be a number between 0 and 2. This property is optional.

bands – the number of bands for spiral, spiral2, exponentialspiral and concentric gradient styles. bands must be a number between 0 and 20. This property is optional.

offsetx – percentage from left to put centre of gradient for centre, circular, radial, spiral, spiral2, exponentialspiral and concentric gradient styles. offsetx must be a number between 0 and 100. This property is optional.

offsety – percentage from top to put centre of gradient for centre, circular, radial, spiral, spiral2, exponentialspiral and concentric gradient styles. offsety must be a number between 0 and 100. This property is optional.

startsize – the start size of the bands in spiral, spiral2, exponentialspiral and concentric gradient styles. startsize must be a number between 1 and 100. This property is optional. Note:

All properties are optional and if they are not set, the SetBackground function will leave the property unchanged.

Remarks:

This function allows you to create the Background style and colour for a specified object by creating a new object and setting the properties you want to change. The new object containing your changes is entered in the objectName parameter and then applied to a specified object.

There are two methods of using this function:

(i) Specify an object name – in this method, the Background

style will be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Background style is set for the Object State of the object specified in the GetAppearance function. Note:

The Background style can be removed using the RemoveBackground function or in the Background tab of the object's Properties dialog.

Example 1:

To set the Background style on the Normal Object State for a Text object named Intro to solid with an opacity of 50% and a colour of blue, use the following syntax:

```
var new_style = new Object() new_style.style = "solid"
new_style.colour = RGB(0,0,255) Intro.SetBackground(new_style)
```

Example 2:

To set the same Background style as Example 1 above on the Mouse Over Object State for a Text object named Intro, use the following syntax:

```
var new_style = new Object() new_style.style = "solid" new_style.colour =
RGB(0,0,255) createEffect = Intro.GetAppearance("Mouse Over")
createEffect.SetBackground(new_style)
```

Example 3:

To set the Background style for a Text object named Intro to a 3 colour spiral (leaving the bands and offset as they were), use the following syntax:

```
var new_style = new Object(); new_style.style = "spiral"; new_style.colour =
new Array(); new_style.colour[0] = RGB(128, 0, 0); new_style.colour[1] =
RGB(0, 128, 0); new_style.colour[2] = RGB(0, 0, 128); new_style.twist = 0.2;
Intro.SetBackground(new_style); Note:
```

In Example 3, the colours are created using an array.

Example 4:

To rotate the angle of a background by 5 degrees, use the following syntax:

```
var new_style = new Object(); new_style.angle = 0; while (true) {
Frame_2.SetBackground(new_style); new_style.angle += 5; wait(0.1); }
```

SetBorder

Syntax:

SetBorder(Style, Round, Width, Radius, Colour1, Colour2, Opacity)

Parameters:

Style – the name of one of the Border styles you can create with this function. Style must be one of the following keywords: None, Plain, Double, Neon, Ellipse, Mask, Raised or Sunken. This parameter is required. The keyword must be surrounded by quote marks or a variable containing one of the valid keywords.

Round – set if the border should have rounded edges. Round is either true to set rounded corners or false to not set rounded corners. This parameter is optional and the default is false.

Width – set the width of the border in pixels. Width is an integer. This parameter is optional and the default is 0.

Radius – set the corner radius i.e. the amount of pixels by which the corner is rounded. The radius is an integer. This parameter is optional and the default is 0.

Colour1 – the first border colour to be used. Colour1 must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is 0.

Colour2 – the second border colour to be used. Colour2 must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is 0.

Opacity – a percentage, showing the level of opacity for the border. Opacity must be an integer between 0 and 100. This parameter is optional and the default is 0. Note:

Different styles use a different amount of colours. For example, Plain only uses one colour, so you only need to enter the Colour1

parameter; Neon requires two colours, so you will need to set values for both Colour1 and Colour2. The RGB value can be calculated using the RGB function.

Remarks:

This function allows you to create the Border style for a specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Border style will

be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Border style is set for the Object State of the object specified in the GetAppearance function. Note:

The Border style can be removed using the RemoveBorderfunction or in the Border tab of the object's Properties dialog.

Example:

Example 1:

To set the Border style on the Normal Object State for a Text object named Intro to plain with a border width of 4 pixels, rounded corners of 10 pixels and a border colour of blue, use the following syntax:

```
MyColour = RGB(0,0,255) Intro.SetBorder("Plain",true,4,10,MyColour)
```

Example 2:

To set the same Border style as Example 1 above on the Mouse Over Object State for a Text object named Intro, use the following syntax:

```
MyColour = RGB(0,0,255) createEffect = Intro.GetAppearance("Mouse Over")
createEffect.SetBorder("Plain",true,4,10,MyColour)
```

SetBtnColour

Syntax:

SetBtnColour(SurfaceColour, LightBevelColour, DarkBevelColour, BevelWidth, BevelOpacity, BlackOutline)

Parameters:

SurfaceColour – the colour of the surface of the button. SurfaceColour must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is RGB(192,192,192) (i.e. grey or string colour "c0c0c0").

LightBevelColour – the light bevel colour of the button. LightBevelColour must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is RGB(255,255,255) (i.e. white or string colour "ffffff").

DarkBevelColour – the dark bevel colour of the button. DarkBevelColour must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is RGB(0,0,0) (i.e. black or string colour "000000").

BevelWidth – set the bevel width of the button in pixels. BevelWidth is an integer. This parameter is optional and the default is 4.

BevelOpacity – a percentage, showing the level of opacity for the border. BevelOpacity must be an integer between 0 and 100. This parameter is optional and the default is 75.

BlackOutline – an outline border of black around the button. BlackOutline is either true if you want a black outline or false if you do not. This paramter is optional and the default is true. Note:

The RGB value can be calculated using the RGB function.

Remarks:

This function allows you to create the Button style for a specified Button object. There are two methods of using this function:

(i) Specify an object name – in this method, the Button style will

be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Button style is set for the Object State of the object specified in the GetAppearance function. Note:

The Button style can be removed using the RemoveBtnColour function or in the Button tab of the object's Properties dialog.

Example 1:

To set the Button style on the Normal Object State for a Button object named Option1 to a surface colour of green, use the following syntax:

```
MyColour = RGB(0,255,0) Option1.SetBtnColour(MyColour)
```

Example 2:

To set the same Button style as Example 1 above on the Mouse Over Object State for a Button object named Option1, use the following syntax:

```
MyColour = RGB(0,255,0) createEffect = Option1.GetAppearance("Mouse Over")
createEffect.SetBtnColour(MyColour)
```

SetFlare

Syntax:

SetFlare(Height, Width, Transparency, Colour, Blur)

Parameters:

Height – the height in pixels of the flare. Height must be a positive integer. This parameter is optional and the default is 3.

Width – the width in pixels of the flare. Width must be a positive integer. This parameter is optional and the default is 3.

Transparency – a percentage, showing the level of transparency for the flare. Transparency must be a positive integer between 0 and 100. This parameter is optional and the default is 50.

Colour – the colour of the flare. Colour must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is 0 (i.e. black).

Blur – the blur value in pixels for the flare. Blur must be an integer between 0 and 5. This parameter is optional and the default is 3. Note:

The RGB value can be calculated using the RGB function.

Remarks:

This function allows you to create the Flare effect for a specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Flare effect will

be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Flare effect is set for the Object State of the object specified in the GetAppearance function. Note:

The Flare effect can be removed using the RemoveFlare function or in the Effects tab of the object's Properties dialog.

Example 1:

To set the Flare effect on the Normal Object State for a Button object named Option1 with a flare height of 5 pixels and width of 3 pixels, the colour yellow, a 50% transparency and a blur of 3 pixels, use the following syntax:

```
MyColour = RGB(253,250,0) Option1.SetFlare(5,3,50,MyColour,3)
```

Example 2:

To set the same Flare effect as Example 1 above on the Mouse Over Object State for a Button object named Option1, use the following syntax:

```
MyColour = RGB(253,250,0) createEffect = Option1.GetAppearance("Mouse Over")
createEffect.SetFlare(5,3,50,MyColour,3)
```

SetImage

Syntax:

SetImage (Filename)

Parameters:

Filename – The filename of the image. Filename should be the full pathname of the image file or the alias name given to the image in the Additional Resources tab of the page or Publication Properties dialog. This parameter is required.

Note:

Pathnames normally contain backslashes e.g. D:\IntroGraph.bmp, all single backslashes should be entered as double backslashes i.e. D:\\IntroGraph.bmp.

Remarks:

This function allows you to inset an Image for a specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Image will be

set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Image is set for the Object State of the object specified in the GetAppearance function. Note:

The Image can be removed using the RemoveImage function or in the Image tab of the object's Properties dialog.

Example 1:

To insert a new Image named Logo.bmp on the Normal Object State for an Image object named IntroGraph, use the following syntax:

```
ImagePath = <PUBLICATION_DIR> + "\\Logo.bmp" IntroGraph.SetImage(ImagePath)
Note:
```

In Example 1 above, the pathname has been created using the System variable <SYSTEM_PUBLICATION_DIR> - this variable contains the pathname to the folder in which the publication is currently running. The name of the image after <SYSTEM_PUBLICATION_DIR> indicates that the image is stored in the same folder as the publication.

Example 2:

To set the same Image as Example 1 above on the Mouse Over Object State for a Button object named Option1, use the following syntax:

```
ImagePath = SYSTEM_PUBLICATION_DIR + "\\Logo.bmp" newImage =
IntroGraph.GetAppearance("Mouse Over") newImage.SetImage(ImagePath)
```

SetShadow

Syntax:

SetShadow(Height, Width, Transparency, Colour, Blur)

Parameters:

Height – the height in pixels of the shadow. Height must be a positive integer. This parameter is optional and the default is 10.

Width – the width in pixels of the shadow. Width must be a positive integer. This parameter is optional and the default is 10.

Transparency – a percentage, showing the level of transparency for the shadow. Transparency must be a positive integer between 0 and 100. This parameter is optional and the default is 75.

Colour – the colour of the shadow. Colour must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is 0 (i.e. black).

Blur – the blur value in pixels for the shadow. Blur must be an integer between 0 and 5. This parameter is optional and the default is 2.

Note:

The RGB value can be calculated using the RGB function.

Remarks:

This function allows you to set the Shadow effect for a specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Shadow effect

will be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearance function – in this method, the Shadow effect is set for the Object State of the object specified in the GetAppearance function. Note:

The Shadow effect can be removed using the RemoveShadow function or in the Effects tab of the object's Properties dialog.

Example 1:

To set the Shadow effect on the Normal Object State for a Button object named Option1 with a shadow height of 5 pixels and width of 3 pixels, the colour yellow, a 50% transparency and a blur of 3 pixels, use the following syntax:

```
MyColour = RGB(253,250,0) Option1.SetShadow(5,3,50,MyColour,3)
```

Example 2:

To set the same Shadow effect as Example 1 above on the Mouse Over Object State for a Button object named Option1, use the following syntax:

```
MyColour = RGB(253,250,0) createEffect = Option1.GetAppearance("Mouse Over")
createEffect.SetShadow(5,3,50,MyColour,3)
```

SetTexture

Syntax:

SetTexture(Gloss, Invert, Animated, Image Path, Tile)

Parameters:

Gloss – the name of one of the gloss styles you can create with this function. Gloss must be one of the following keywords: Paper, Card, Silk, Plastic, Metal or Glass. Alternatively you can enter an integer between 0 and 20. This parameter is optional and the default is Plastic. The keyword must be surrounded by quote marks or a variable containing one of the valid keywords.

Invert – set if the texture effect should be inverted or not. Invert is either true if you want to invert the texture or false if not. This parameter is optional and the default value is false.

Animated – set if the Image used is animated or not. Animated is either true if the image used is animated (e.g. an animated gif) and false if not. This parameter is optional and the default value is false.

Image Path – The filename of the image. Image Path should be the full pathname of the image file or the alias name given to the image in the Additional Resources tab of the page or Publication Properties dialog. This parameter is optional.

Note:

Pathnames normally contain backslashes e.g. D:\IntroGraph.bmp, all single backslashes should be entered as double backslashes i.e. D:\\IntroGraph.bmp.

Tile – set if the image used in Image Path should be tiled across the surface of the object or not. Tile is either true if the image should be tiled (i.e. repeated across the surface of the object until the object is filled) or false if the image is used once and stretched over the surface of the object. This parameter is optional and the default value is false.

Remarks:

This function allows you to create the Texture effect for a specified object. There are two methods of using this function:

(i) Specify an object name – in this method, the Texture effect

will be set on the Normal (i.e. default) Object State.

(ii) Specify a new object created with the GetAppearancefunction – in this method, the Texture effect is set for the Object State of the object specified in the GetAppearance function. Note:

The Texture effect can be removed using the RemoveTexture function or in the Effects tab of the object's Properties dialog.

Example 1:

To set the Texture effect on the Normal Object State for an Image object named IntroGraph with a Silk gloss and an image file named CompLogo.bmp that is stretched over the surface of the Image object, use the following syntax:

```
ImageUsed = SYSTEM_PUBLICATION_DIR + "\\CompLogo.bmp"
IntroGraph.SetTexture("Silk",false,false,ImageUsed,fa lse)
```

Note:

In Example 1 above, the variable ImageUsed contains the pathname to the image (CompLogo.bmp) that will be used to create the texture – the <SYSTEM_PUBLICATION_DIR> is a System variable that contains the pathname of the current publication.

Example 2:

To set the same Texture effect as Example 1 above on the Mouse Over Object State for an Image object named IntroGraph, use the following syntax:

```
ImageUsed = SYSTEM_PUBLICATION_DIR + "\\CompLogo.bmp " createEffect =
IntroGraph.GetAppearance("Mouse Over")
createEffect.SetTexture("Silk",false,false,ImageUsed)
```

CaptureMouse

Syntax:

CaptureMouse()

Remarks:

This function causes all mouse input to got to the specified object only, until either the ReleaseMouse function is called or another object captures the mouse. For example, if an object has captured the mouse input, actions such as left-click, right-click or rollover will have no effect until the ReleaseMouse function is set. No parameters are required.

Example:

For a Button object named grabMouse, use the following syntax in its Script Action.

```
grabMouse.CaptureMouse()
```

ReleaseMouse

Syntax:

ReleaseMouse()

Remarks:

This function will return the mouse input handling to normal, if the specified object had used a CaptureMouse function e.g. the mouse will execute left-click, right-click or rollover actions as normal.

Example:

For a Button object named grabMouse, use the following syntax to release the mouse capture

```
grabMouse.ReleaseMouse()
```

SetFocus

Syntax:

SetFocus()

Remarks:

This function causes all keyboard input to go to this object only.

Example:

For a Text Input object named answer, use the following syntax:

```
answer.SetFocus()
```

GetScrollInfo

Syntax:

GetScrollInfo()

Return:

A new object with the following properties:

vscrollbar - the vertical scrollbar object, if any.

vvisible - the percentage of the specified object visible vertically.

vpos - the vertical scroll position of the specified object as a percentage.

hscrollbar - the horizontal scrollbar object, if any.

hvisible - the percentage of the specified object visible horizontally.

hpos - the horizontal scroll position of the specified object as a percentage.

Remarks:

This function gets the scroll position, scroll size and name of the scrollbar attached to an Object. The Return Value of the function is a new object containing the properties listed above.

Example:

For a Text object named myText, to get the scrollbar information, use the following syntax:

```
var scrollInfo = myText.GetScrollInfo()
```

Note: In this example, the variable scrollInfo becomes a new object containing the Return Value properties of myText. To reference a property, use the following syntax:

```
scrollInfo.vpos //the vertical scroll position of the object.
ScrollInfo.vscrollbar //the vertical scrollbar object.
```

SetScrollPosition

Syntax:

SetScrollPosition(VScrollPos, HScrollPos)

Parameters:

VScrollPos - The vertical scroll position as a percentage. If VScrollPos is –1, the vertical position remains as is. This parameter is required.

HScrollPos - The horizontal scroll position as a percentage. If HScrollPos is –1, the horizontal position remains as is. This parameter is required.

Remarks:

This function allows you to set the position of the scrollbar for a specified Object. Both parameters are required. The parameters can either be a percentage or a variable name containing a percentage value.

Example:

For example, to set the scrollbar position of Text object text2 to exactly the same position as Text object text1, use the following syntax:

```
var scrollPos = text1.GetScrollInfo()
text2.SetScrollPosition(scrollPos.vpos,scrollPos.hpos )
```

SetPosition

Syntax:

SetPosition(PosX, PosY, Time, Wait)

Parameters:

PosX – the horizontal coordintate of the specified object in pixels. PosX must be an integer. This parameter is required.

PosY – the vertical coordinate of the specified object in pixels. PosY must be an integer. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the x and y coordinates of the specified Object's mid point, relative to the top left-hand corner of its containing frame or to its page if it is not in a frame. There is also an optional Time parameter.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetPosition(30,15,2.784)
```

SetPositionX

Syntax:

SetPositionX(PosX, Time, Wait)

Parameters:

PosX – the horizontal coordinate of the specified object in pixels. PosX must be an integer. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or

not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function moves the specified object horizontally in its frame to a position PosX pixels from the left edge of its container frame (or page if it is not in a frame). The y coordinate is left unchanged by this function. There is also an optional Time parameter.

Note: The actual motion is dependent upon the parents' frame of reference i.e. if the parent frame is rotated by 45 degrees, then the motion of the specified object would also be rotated by 45 degrees.

Example:

For an Image object named myImage, to change its horizontal position, use the following syntax:

```
myImage.SetPositionX(100,2.0)
```

SetPositionY

Syntax:

SetPositionY(PosY, Time, Wait)

Parameters:

PosY – the vertical coordinate of the specified object in pixels. PosY must be an integer. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function moves the specified Objects mid point vertically in its frame to a position PosY pixels from top edge of its container frame (or page if not in a frame). The x coordinate is left unchanged by this function. There is also an optional Time parameter.

Note: The actual motion is dependent upon the parents' frame of reference i.e. if the parent frame is rotated by 45 degrees, then the motion of the specified object would also be rotated by 45 degrees.

Example:

For an Image object named myImage, to change its vertical position, use the following syntax:

```
myImage.SetPositionY(50,2.0)
```

Move

Syntax:

Move(PosX, PosY, Time, Wait)

Parameters:

PosX – PosX is the amount in pixels by which to move the centre of the object in the x direction. PosX can be positive or negative e.g. (20 or –20). This parameter is required.

PosY – PosY is the amount in pixels by which to move the centre of the object in the y direction. PosY can be positive or negative e.g. (20 or –20). This parameter is required.

Time - A length of time in seconds to animate the move over. This parameter is optional.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function will move the specified object by the number of pixels entered in PosX and PosY. If Time is not specified, the move occurs immediately. If the specified object is inside a frame then the move will be with respect to the top left corner of the frame and not the page.

Example:

For an Image object named myImage to move 20 pixels down and 30 pixels across in half a second, use the following syntax:

```
myImage.Move(20,30,0.5)
```

MoveX

Syntax:

MoveX(PosX, Time, Wait)

Parameters:

PosX – PosX is the amount in pixels by which to move the centre of the object in the x direction. PosX can be positive or negative e.g. (20 or –20). This parameter is required.

Time - A length of time in seconds to animate the move over. This parameter is optional.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function moves the specified Object's mid point by PosX pixels across its container frame (or page if it is not in a frame). The y coordinate is left unchanged by this function. There is an

optional Time parameter – see Remarks in function Move).

Example:

For an Image object named myImage use the following syntax:

```
myImage.MoveX(50, 2.0)
```

MoveY

Syntax:

MoveY(PosY, Time, Wait)

Parameters:

PosY – PosY is the amount in pixels by which to move the centre of the object in the y direction. PosY can be positive or negative e.g. (20 or –20). This parameter is required.

Time - A length of time in seconds to animate the move over. This parameter is optional.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function moves the specified object's mid point by PosY pixels down its container frame (or page if it is not in a frame). The y coordinate is left unchanged by this function. There is an optional Time parameter – see Remarks in function Move).

Example:

For an Image object named myImage use the following syntax:

```
myImage.MoveY(50, 2.0)
```

SetTransparency

Syntax:

SetTransparency(Trans, Time, Wait)

Parameters:

Trans – set the transparency of the specified Object. Trans should be a percentage ranging from 0 = opaque and 100 = transparent. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetTransparency(50,1.0)
```

Fade

Syntax:

Fade(Trans, Time, Wait)

Parameters:

Trans – set the transparency of the specified Object. Trans should be a percentage. If Trans is a negative number (e.g.-50) it is a fade-in. If Trans is a positive number (e.g.75) it is a fade-out. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait

is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function adjusts the current transparency of the specified object by the amount in Trans. The value of Trans is added to the current transparency of the specified Object. There is an optional Time facility.

Example:

For an Image object named myImage, with an initial transparency of 50%, use the following syntax:

```
myImage.Fade(25,1.0) // after 1 second the transparency of myImage // is 75%
```

```
i.e. 50 + 25. MyImage.Fade(-50,1.0) // after 1 second the transparency of
myImage // is 25% i.e. 75 - 50.
```

SetRotation

Syntax:

SetRotation(Angle, Direction, Time, Wait)

Parameters:

Angle – set the degree of rotation of the specified Object. Angle should be an integer. This parameter is required.

Direction – set the direction of rotation. Direction is either true or false. If Direction is true the specified object is rotated clockwise. If Direction is false the specified object is rotated anti-clockwise. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is

completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects rotation to the amount set in Angle and Direction over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetRotation(45,true,2.0)
```

Note:

In this example, if myImage.SetRotation were called more than once, myImage would remain at 45%.

Rotate

Syntax:

Rotate(Angle,Time, Wait)

Parameters:

Angle – set the degree of rotation of the specified Object. Angle should be an integer. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects rotation to the amount set in Angle over a specified

time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.Rotate(45,2.0) Note:
```

In this example, if myImage.Rotate(45,2.0) was called once, the image will rotate 45 degrees from its current position. If called again, the image will rotate another 45 degrees.

SetRoll

Syntax:

SetRoll(Angle, Direction, Time, Wait)

Parameters:

Angle – set the degree of rotation of the specified Object. Angle should be an integer. This parameter is required.

Direction – set the direction of rotation. Direction is either true or false. If Direction is true the specified object is rolled forwards. If Direction is false the specified object is rolled backwards. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects orientation on the x-axis to the amount set in Angle and Direction over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetRoll(45,true,2.0)
```

Roll

Syntax:

Roll(Angle, Time, Wait)

Parameters:

Angle – The angle to rotate the object by in degrees. If Angle is positive (e.g. 90), the object is rotated in a clockwise direction around the x-axis. If Angle is negative, the object is rotated in an anti-clockwise direction around the x-axis. This pararemter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function will roll the specified object by the set angle over a time set in seconds. Every time you use this function, it will rotate the same amount set in Angle again.

Example:

For an Image object named myImage use the following syntax:

```
myImage.Roll(45,2)
```

SetSpin

Syntax:

SetSpin(Angle, Direction, Time, Wait)

Parameters:

Angle – set the degree of rotation of the specified object. Angle should be an integer. This parameter is required.

Direction – set the direction of spin. Direction is either true or false. If Direction is true the specified object spins right. If Direction is false the specified object spins left. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified objects orientation on the y-axis to the amount set in Angle and Direction over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetSpin(90,true,2.0)
```

Spin

Syntax:

Spin(Angle, Time, Wait)

Parameters:

Angle – set the degree of rotation of the specified object from its current position. Angle should be an integer. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects orientation on the y-axis to the amount set in Angle and Direction over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.Spin(90,2.0)
```

Note: In this example, if myImage.Spin(90,2.0) was called once, the image will spin 90 degrees from its current position. If called again, the image will scale another 90 degrees.

SetSkew

Syntax:

SetSkew(Horizontal, Vertical, Time, Wait)

Parameters:

Horizontal – set the percentage of horizontal skew of the specified Object. Horizontal should be a number where 0 = no skew and 100 = full skew. This parameter is required.

Vertical – set the percentage of vertical skew of the specified Object. Vertical should be a number where 0 = no skew and 100 = full skew. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects horizontal and vertical skew by the amount set in Horizontal and Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetSkew(35, 45,2.0)
```

Note: In this example, if myImage.SetSkew(35,45,2.0) were called more than once, myImage would remain at 35%.

SetSkewH

Syntax:

SetSkewH(Horizontal, Time, Wait)

Parameters:

Horizontal – set the percentage of horizontal skew of the specified Object. Horizontal should be a number where 0 = no skew and 100 = full skew. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects horizontal skew by the amount set in Horizontal over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetSkewH(35,2.0)
```

Note: In this example, if myImage.SetSkewH(35,2.0) were called more than once, myImage would remain at 35%.

SetSkewV

Syntax:

SetSkewV(Vertical, Time, Wait)

Parameters:

Vertical – set the percentage of vertical skew of the specified Object. Vertical should be a number where 0 = no skew and 100 = full skew. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is

completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects vertical skew by the amount set in Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetSkewV(45,2.0)
```

Note: In this example, if myImage.SetSkewV(45,2.0) were called more than once, myImage would remain at 45%.

Skew

Syntax:

Skew(Horizontal, Vertical, Time, Wait)

Parameters:

Horizontal – set the percentage of horizontal skew of the specified object from its current skew position. If Horizontal is a positive number it is added to the current skew value. If Horizontal is a negative number it is subtracted from the current skew value. This parameter is required.

Vertical – set the percentage of vertical skew of the specified object from its current skew position. If Vertical is a positive number it is added to the current skew value. If Vertical is a negative number it is subtracted from the current skew value. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function rotates the specified Objects horizontal and vertical skew by the amount set in Horizontal and Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.Skew(35,45,2.0)
```

Note:

In this example, if myImage.Skew(35,45,2.0) was called once, the image will skew 35% horizontally and 45% vertically from its current position. If called again, the image will skew another 35% and 45% respectively.

SkewH

Syntax:

SkewH(Horizontal, Time, Wait)

Parameters:

Horizontal – set the percentage of horizontal skew of the specified object from its current skew position. If Horizontal is a positive number it is added to the current skew value. If Horizontal is a negative number it is subtracted from the current skew value. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function rotates the specified Objects horizontal skew by the amount set in Horizontal over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SkewH(35,2.0)
```

Note:

In this example, if myImage.SetSkewH(35,2.0) was called once, the image will skew 35% horizontally from its current position. If called again, the image will skew another 35%.

SkewV

Syntax:

SkewV(Vertical, Time, Wait)

Parameters:

Vertical – set the percentage of vertical skew of the specified object from its current skew position. If Vertical is a positive number it is added to the current skew value. If Vertical is a negative number it is subtracted from the current skew value. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function rotates the specified Objects vertical skew by the amount set in Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SkewV(45,2.0) Note:
```

In this example, if myImage.SetSkewV(45,2.0) was called once, the image will skew 45% vertically from its current position. If called again, the image will skew another 45%.

SetScale

Syntax:

SetScale(Horizontal, Vertical, Time, Wait)

Parameters:

Horizontal – Enter a number (the scale factor) to scale the horizontal size of the specified object. If the number of Horizontal is 2.0, the specified objects horizontal size will be 200% larger than the original size, while the number 0.5 will make it 50% of the original size. This parameter is required.

Vertical – Ienter a number (the scale factor) to scale the vertical size of the specified object. If the number of Vertical is 2.0, the specified object will be 200% larger than the original size, while the number 0.5 will make it 50% of the original size. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects horizontal and vertical scale to the amount set in Horizontal and Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetScale(2,2,2.0) Note:
```

In this example, myImage will be 200% larger than the original size, for example, if the horizontal and vertical size were both 40 pixels, the new values would be 80 pixels. If myImage.SetScale(2,2,2.0) was called more than once, myImage will remain the same size as the first time this function was called because 200% of the original size will always return the same result. Entering a different scale factor, for example, SetScale(3,3,1.0) would scale it to 300% its original size. If you want to constantly increase the size of an object by a particular

scale factor, use the Scale functions.

SetScaleH

Syntax:

SetScaleH(Horizontal, Time, Wait)

Parameters:

Horizontal – Enter a number (the scale factor) to scale the horizontal size of the specified object. If the number of Horizontal is 2.0, the specified objects horizontal size will be 200% larger than the original size, while the number 0.5 will make it 50% of the original size. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects horizontal scale by the amount set in Horizontal over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetScaleH(2,2.0) Note:
```

In this example, myImage horizontal size will be 200% larger than the original size, for example, if the horizontal size was 40 pixels, the new values would be 80 pixels. If myImage.SetScaleH(2,2.0) was called more than once, myImage will remain the same size as the first time this function was called because 200% of the original size will always return the same result. Entering a different scale factor, for example, SetScaleH(3,1.0) would scale the horizontal size to 300% its original size. If you want to constantly increase the size of an object by a particular scale factor, use the Scale functions.

SetScaleV

Syntax:

SetScaleV(Vertical, Time, Wait)

Parameters:

Vertical – Enter a number (the scale factor) to scale the vertical size of the specified object. If the number of Vertical is 2.0, the specified object will be 200% larger than the original size, while the number 0.5 will make it 50% of the original size. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects vertical scale by the amount set in Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetScaleV(2,2.0) Note:
```

In this example, myImage vertical size will be 200% larger than the original size, for example, if the vertical size was 40 pixels, the new values would be 80 pixels. If myImage.SetScaleV(2,2.0) was called more than once, myImage will remain the same size as the first time this function was called because 200% of the original size will always return the same result. Entering a different scale factor, for example, SetScaleV(3,1.0) would scale the vertical size to 300% its original size. If you want to constantly increase the size of an object by a particular scale factor, use the Scale functions.

Scale

Syntax:

Scale(Horizontal, Vertical, Time, Wait)

Parameters:

Horizontal – Enter a number to add to the current scale factor for the horizontal size of the specified object. The original scale factor of an object is always 1.0 (i.e. 100% of its own size), with this function the number of Horizontal is added to the scale factor. For example, if Horizontal is 0.5 then the scale factor for the specified object will be 1.5 and the horizontal size will be 150% of its original size. Horizontal can be a positive or negative number. This parameter is required.

Vertical – Enter a number to add to the current scale factor for the vertical size of the specified object (see Horizontal for more information on scale factors). Vertical can be a positive or negative number. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects horizontal and vertical scale by the amount set in

Horizontal and Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetScale(2,2,1.0) // sets the object size to 200% of its original
size myImage.Scale(0.5, 1,2.0) Note:
```

In this example, myImage is initially set to 200% of its original size using the SetScale function. By calling myImage.Scale(0.5,1,2.0), the image will be 2.5 times wider and 3 times taller than its original size. If the function is called again, the image will be 3 times wider and 4 times taller because the effect of this function is cumulative, in other words, the scale factor is added to the current scale factor for the specified object.

If you want to reduce the scale factor by a particular amount then make the Horizontal or Vertical parameters a negative number. For example, to reduce the scale of an object named myImage, which is 200% of its original size, to 150%, use the following:

```
myImage.Scale(-0.5,-0.5,1.0)
```

ScaleH

Syntax:

ScaleH(Horizontal, Time, Wait)

Parameters:

Horizontal – Enter a number to add to the current scale factor for the horizontal size of the specified object. The original scale factor of an object is always 1.0 (i.e. 100% of its own size), with this function the number of Horizontal is added to the scale factor. For example, if Horizontal is 0.5 then the scale factor for the specified object will be 1.5 and the horizontal size will be 150% of its original size. Horizontal can be a positive or negative number. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or

false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects horizontal scale by the amount set in Horizontal over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetScale(2,2,1.0) // sets the object size to 200% of its original
size myImage.ScaleH(0.5,2.0) Note:
```

In this example, myImage is initially set to 200% of its original size using the SetScale function. By calling myImage.ScaleH(0.5,2.0), the image will be 2.5 times wider than its original size. If the function is called again, the image will be 3 times wider because the effect of this function is cumulative, in other words, the scale factor is added to the current scale factor for the specified object.

If you want to reduce the scale factor by a particular amount then make the Horizontal parameter a negative number. For example, to reduce the horizontal scale of an object named myImage, which is 200% of its original size, to 150%, use the following:

```
myImage.ScaleH(-0.5,1.0)
```

ScaleV

Syntax:

ScaleV(Vertical, Time, Wait)

Parameters:

Vertical – Enter a number to add to the current scale factor for the vertical size of the specified object. The original scale factor of an object is always 1.0 (i.e. 100% of its own size), with this function the number of Vertical is added to the scale factor. For example, if Vertical is 0.5 then the scale factor for the specified object will be 1.5 and the vertical size will be 150% of its original size. Vertical can be a positive or negative number. This parameter is required.

Time – the time period in seconds. This parameter is optional. The default value is 0.0 seconds

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function sets the specified Objects vertical scale by the amount set in Vertical over a specified time. The Time parameter is optional.

Example:

For an Image object named myImage use the following syntax:

```
myImage.SetScale(2,2,1.0) // sets the object size to 200% of its original
size myImage.ScaleV(0.5,2.0) Note:
```

In this example, myImage is initially set to 200% of its original size using the SetScale function. By calling myImage.ScaleV(0.5,2.0), the image will be 2.5 times taller than its original size. If the function is called again, the image will be 3 times taller because the effect of this function is cumulative, in other words, the scale factor is added to the current scale factor for the specified object.

If you want to reduce the scale factor by a particular amount then make the Vertical parameter a negative number. For example, to reduce the vertical scale of an object named myImage, which is 200% of its original size, to 150%, use the following:

```
myImage.ScaleV(-0.5,1.0)
```

StopAnimation

Syntax:

StopAnimation(StopCode, Wait)

Parameters:

StopCode – is a string containing one or more of the following options (each option should be separated by a "|" character. StopCode must be surrounded by quote marks. This parameter is required:

Roll – stop object rolling.

Spin – stop object spinning.

Rotate – stop object rotating.

Scale – stop horizontal and vertical scaling.

ScaleH – stop horizontal scaling.

ScaleV – stop vertical scaling.

Skew – stop horizontal and vertical skewing.

SkewH – stop horizontal skewing.

SkewV – stop vertical skewing.

Move – stop all x and y movements.

MoveX – stop all x movement.

MoveY – stop all y movement.

Fade – stop fading.

Path – stop x and y motion, plus rotation i.e. the types of animation undertaken when an object follows a path.

ALL – stop all current animations.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function stops the animations selected within StopCode for the specified Object.

Example:

For an Image object named myImage use the following syntax:

```
myImage.StopAnimation("Roll|Spin|ScaleH|Skew") Note:
```

In this example, all roll, spin, horizontal scaling and all skewing actions on the object myImage are stopped at once.

ResetAnimation

Syntax:

ResetAnimation(StopCode, Time, Wait)

Parameters:

StopCode – is a string containing one or more of the following options (each option should be separated by a "|" character. StopCode must be surrounded by quote marks. This parameter is required:

Roll – stop object rolling.

Spin – stop object spinning.

Rotate – stop object rotating.

Scale – stop horizontal and vertical scaling.

ScaleH – stop horizontal scaling.

ScaleV – stop vertical scaling.

Skew – stop horizontal and vertical skewing.

SkewH – stop horizontal skewing.

SkewV – stop vertical skewing.

Move – stop all x and y movements.

MoveX – stop all x movement.

MoveY – stop all y movement.

Fade – stop fading.

Path – stop x and y motion, plus rotation i.e. the types of animation undertaken when an object follows a path.

ALL – stop all current animations.

Time – the time over which the reset should occur.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function resets the animations selected within the StopCode for the specified Object.

Example:

For an Image object named myImage use the following syntax:

```
myImage.ResetAnimation("Roll|Spin",1.0) Note:
```

In this example, all roll and spin actions on the object myImage are reset over 1 second.

FollowPath

Syntax:

FollowPath(Path, Time, Relative, Orientation, Start, End, Wait)

Parameters:

Path – The path object to follow. This parameter is required.

Time – The number of seconds to take when following the path. Time should be a number. This parameter is optional. The default value is 0.0 seconds.

Relative – The motion of the specified Object. Relative is either true or false. If Relative is true, the motion is relative to the Objects current position. If Relative is false, the motion follows the path absolutely (i.e. the object will "jump" to the position of the path object). This parameter is optional. The default value is true.

Orientation – The orientation of the specified object along the path. Orientation is either true or false. If Orientation is true, the Object's orientation is aligned to the path. If Orientation is false, the object orientation is fixed. This parameter is optional. The default value is false.

Start – The start position for the specified object along Path. Start is a percentage between 0 and 100. This parameter is optional. The default value is 0.

End – The end position for the specified object along Path. End is a percentage between 0 and 100. This parameter is optional. The default value is 100.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait

is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function animates the specified object along Path, over a period Time from positions on the path set in Start and End. The specified Object's orientation and motion can also be set with this function.

Example:

For an Image object named myImage, use the following syntax:

```
myImage.FollowPath(Path1, 2.0, false, true, 10, 90, true) myImage.Hide()
```

Note:

In this example, myImage will follow the path named Path1, it will take two seconds for the animation to complete, myImage follows the path relative to it's current location and its orientation is aligned to the path. The animation starts 10% in from the beginning of the path and finishes 90% along the path. Finally, the script will wait for this animation to complete before it processes the next line of script that hides myImage.

GetTotalLength

Syntax:

GetTotalLength()

Return:

The total length of an Animation Path object. The value is in squared pixel distance, in other words, for a 10 pixel horizontal path, the return value is 100.

Remarks:

This function will return the length of a specified Animation Path object. No parameters required.

Example:

For an Animation Path object named Path 1, use the following syntax:

```
var pathLength = Path_1.GetTotalLength()
```

GetPositionFromPercent

Syntax:

GetPositionFromPercent(Percent)

Return:

A new object with two properties: the x co-ordinate of the entered percent; and the y co-ordinate of the entered percent – in relation to the specified animation path.

Parameters:

Percent – set the percentage distance along the Animation Path for which you want to return the x and y co-ordinates. Percent must be a number between 0 (start of path) and 100 (end of path). This parameter is required.

Remarks:

This function will return a new object containing an x and y co- ordinate for Percent.

Example:

For an Animation Path object named Path 1, use the following syntax:

```
var percent = Path_1.GetPositionFromPercent() var PosX = percent.x var PosY = percent.y
```

GetPosition

Syntax:

GetPosition(Length)

Return:

A new object with two properties: the x co-ordinate of the entered length; and the y co-ordinate of the entered length – in relation to the specified animation path.

Parameters:

Length – set the length along the animation path for which you want to return the x and y co-ordinates. Length is the number of square pixel units along the specified animation path Object. This parameter is required.

Remarks:

This function will return a new object containing an x and y co- ordinate for Length.

Example:

For an Animation Path object named Path 1, use the following syntax:

```
var length = Path_1.GetPosition(34) var PosX = length.x var PosY = length.y
```

CloneObject

Syntax:

CloneObject(PosX, PosY, Visible)

Parameters:

PosX – the x-coordinate of the cloned object. If PosX is specified, then you must also specify a value for the PosY parameter as well. This parameter is optional and if no value is set, the cloned object will be positioned over the original object.

PosY – the y-coordinate of the cloned object. If PosY is specified, then you must also specify a value for the PosX parameter as well. This parameter is optional and if no value is set, the cloned object will be positioned over the original object.

Visible – used to show or hide the cloned object. Visible can be set to true or false. If true, the cloned object is shown on screen and if false, the cloned object is not visible. This parameter is optional and if no value is set then the cloned object will be set to the same state as the object being cloned i.e. if the original object is hidden then the cloned object is hidden; if original object is displayed then so is the cloned object.

Return:

The new clone of the object.

Remarks:

This function makes an identical copy of the specified object and returns this new clone. The clone will appear on the page over the top of the original object unless you have set the PosX

and PosY coordinates.

Example:

To clone the Image object myImage and display the cloned object 50 pixels across and 50 pixels down from the top left-hand corner of the page, use the following syntax:

```
var myClone = myImage.CloneObject(50,50)
```

DestroyClonedObject

Syntax:

DestroyClonedObject()

Return:

true if the specified cloned object is deleted. Otherwise, false.

Remarks:

This function will destroy (i.e. delete) a cloned object reference in an OpusScript and the object itself from the page. Only objects that have been created with the CloneObject funciton can be destroyed with this function. No parameters required.

Example:

For a cloned object named myClone, use the following syntax:

```
myClone.DestroyClonedObject()
```

GetLayer

Syntax:

GetLayer()

Return:

The number of the layer, the specified object is on.

Remarks:

This function will find the layer the specified object is currently on. The SetLayer function can change the current layer of an object to another level. No parameters required.

Example:

For an Image object named myImage, use the following syntax:

```
var whatLayer = myImage.GetLayer()
```

SetLayer

Syntax:

SetLayer(Number)

Parameters:

Number – The number of the layer on which you want to place this Object. The number must be an integer. The number is zero- based indexed i.e. the first layer is index position 0, the second layer is index position 1, and so on. This parameter is required.

Remarks:

This function will set the layer position of an Object. This function is useful for making Objects appear above and below other Objects on the screen, for example, if you want to move an object to the front of the screen, you could set the layer number to 0, which is the top layer.

Example:

For an Image object named myImage, use the following syntax:

```
myImage.SetLayer(2)
```

CopyToClipboard

Syntax:

CopyToClipboard(Native)

Parameters:

Native – the format in which to copy a Text object. Native is either true or false. If Native is true it will copy the contents of the Text object in RTF and plain text formats. If Native is false,

the Text object is copied as an image. This parameter is only required when copying Text objects.

Remarks:

This function will copy graphical Objects to the Windows clipboard. A graphical object is any object that can be drawn on a page, it does not include Objects such as Timelines, DocViews and Browsers.

Example:

For an Image object named myImage, use the following syntax:

```
myImage.CopyToClipboard()
```

Pages - Overview

The Page functions are used in a script to manipulate pages in a publication. For example, you can add a page to the Bookmark feature and then show the page you have bookmarked. These functions are equivalent to the Bookmark actions in the Bookmarks menu of the Actions dialog.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Pages

Basic Objects

Functions:

ClearBookmark Clear a Bookmark for the current page

CopyToClipboard Copy the page to the Windows clipboard

GetBookmarkPage Get Bookmark page by its index number

GetFirstBookmark Get the first Bookmark page

GetMousePosition Get the x and y coordinates of the current

mouse position

GetNextBookmark Get the next Bookmark page

GetNextPage Get the next page in the publication

GetPageNumber Return the page number of the current page

open in the publication

GetPreviousPage Get the previous page in the publication

GetPublication Get the name of the current publication

GotoBookmark Go to a specific Bookmark

ResetVars Reset all page properties variables

SetBookmark Bookmark the current page

ShowBookmarkDialog Show the Bookmark Dialog box

GetNextPage

Syntax:

GetNextPage()

Return:

The page following this one in the publication. If this is the last page in a chapter, the first page

in the next chapter is returned. If this is the last page in the publication NULL is returned.

Remarks:

This function gets the page following this one in the publication. No parameters required.

```
Example: var nextPage = GetNextPage()
```

GetPreviousPage

Syntax:

GetPreviousPage()

Return:

The page before this one in the publication. If this is the first page in a chapter, the last page in the previous chapter is returned. If this is the first page in the publication NULL is returned.

Remarks:

This function gets the page before this one in the publication. No parameters required.

Example:

```
var prevPage = GetPreviousPage()
```

GetMousePosition

Syntax:

GetMousePosition()

Return:

A new object with the following properties:

x - the current x coordinate of the mouse with respect to the top left corner of the page.

y - the current y coordinate of the mouse with respect to the top left corner of the page.

Remarks:

This function returns the x and y coordinates of the mouse. No parameters required.

Example:

Example 1:

In this example, the variable mousePos becomes a new object containing the mouse properties.

```
var mousePos = GetMousePosition()
```

Example 2:

To reference the x and y coordinates for mousePos (Example 1 above), use the following syntax:

```
var mouseX = mousePos.x var mouseY = mousePos.y Note:
```

If mouseX and mouseY were page or publication variables, the current mouse position could be displayed on the current page.

GetPublication

Syntax:

GetPublication()

Return:

The publication Object.

Remarks:

This function will return the current publication in which this function was called. No parameters required.

Example:

```
var pubName = GetPublication()
```

ResetVars

Syntax:

ResetVars()

Remarks:

This function will reset all the variables on the current page back to their default values. There are no parameters. This applies only to the variables present in the page's properties, not to any other variables created in the script.

Example:

```
ResetVars()
```

SetBookmark

Syntax:

SetBookmark()

Remarks:

This function will Bookmark the current page in the publication and add it to the list of Bookmarked pages.

```
Example: SetBookmark()
```

ClearBookmark

Syntax:

ClearBookmark()

Remarks:

This function will clear the Bookmark for the current page in the publication and remove the page name from the list of Bookmarked pages.

Example:

```
ClearBookmark()
```

GotoBookmark

Syntax:

GotoBookmark()

Remarks:

This function will go to the nearest bookmark set in this publication.

Example:

```
GotoBookmark()
```

ShowBookmarkDialog

Syntax:

ShowBookmarkDialog()

Remarks:

This function will open the Bookmarks dialog box, which contains a list of all the bookmarked pages in the current publication. The Bookmarks dialog is only available when the publication has been set to use multiple Bookmarks.

Example:

```
ShowBookmarkDialog()
```

GetFirstBookmark

Syntax:

GetFirstBookmark()

Return:

The reference name of the Page object.

Remarks:

This function will return the reference name of the first Page object that has been bookmarked. This function is normally used when a publication has been set to use multiple Bookmarks.

```
Example: var Bmark = GetFirstBookmark()
```

GetNextBookmark

Syntax:

GetNextBookmark()

Return:

The reference name of the Page object.

Remarks:

This function will return the reference name of the next Page object that has been bookmarked. This function is normally used when a publication has been set to use multiple Bookmarks.

```
Example: var Bmark = GetNextBookmark()
```

GetBookmarkPage

Syntax:

GetBookmarkPage(Index)

Return:

The reference name of the Page object.

Parameters:

Index – enter the index number of the page. Index must be an integer or a variable containing an integer. The Index position starts at 1. This parameter is required.

Remarks:

This function will return the reference name of the Page object with the specified index number. This function is normally used when a publication has been set to use multiple Bookmarks.

```
Example: var pageName = GetBookmarkPage(2)
```

CopyToClipboard

Syntax:

CopyToClipboard()

Remarks:

This function will copy an image of the current page to the Windows clipboard.

```
Example: CopyToClipboard()
```

GetPageNumber

Syntax:

GetPageNumber()

Return:

The name of the current page open in the publication.

Remarks:

This function will return the page number of the current page open in the publication. Each page in a publication is numbered consecutively beginning at the number 1. The order is determined by the position they occupy in the Page Organiser running down the left-hand side of the Opus Editor. The first page in the first chapter of a publication is given the number 1, the next page 2, and so on. The number increases by 1 even if your publication is split into chapters and Master Pages are not included in the count.

Example:

In the example below, the variable pageNum will contain the number of the current page open in the publication.

```
var pageNum = GetPageNumber()
```

Publications - Overview

The Publication functions can be used in a script to access information about your publication. For example, you can find out how long the user has been working in your publication using the TimeGetSeconds function. The Publication functions are unique to the Publication.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Publications

Basic Objects

Functions:

GetLastPage Get the name of the last page in the publication

GetPage Get a page by name

GetPublicationKey Get the the evaluation registration key for this publication.

ResetVariables Reset all the publications variables to their default

TestPublicationKey Test a string against the evaluation registration key for this publication.

TimeGetSeconds Get the number of seconds a publication has been running

GetPage

Syntax:

GetPage(Name)

Return:

The page object specified in Name.

Parameters:

Name – The name of the page to return. Name is either the name of the page entered as a string, which should be surrounded in quote marks; or the index position of the page (the pages are zero- based indexed). This parameter is required.

Remarks:

This function gets a page and returns its page object. The page object can be used with other OpusScript functions, such as PrintPage. The parameter Name is used to specify the page either by its string name or by its index position.

Example 1:

To return a page object into a variable named page, use the following syntax:

```
var page = GetPage("page 1") // this gets the page object via the Name var
```

```
page = GetPage(0) // this gets the page via the Index and returns the first
// page in the publication
```

Example 2:

The returned page object can be used with OpusScript functions related to the page. For example, to print the first page in the publication, use the following syntax:

```
var firstPage = GetPage(0) PrintPage(firstPage,true)
```

GetLastPage

Syntax:

GetLastPage()

Return:

The page object of the last page in the publication.

Remarks:

This function returns a page object for the last page in the publication. No parameters required.

Example 1:

To return the page object into a variable named page for the last page in your publication, use the following syntax:

```
var lastPage = GetLastPage()
```

Example 2:

The returned page object can be used with OpusScript functions related to the page. For example, to print the last page in the publication, use the following syntax:

```
var lastPage = GetLastPage() PrintPage(lastPage,true)
```

ResetVariables

Syntax:

ResetVariables()

Remarks:

This function will reset all of the Publication variables back to their default values.

```
Example: ResetVariables()
```

TimeGetSeconds

Syntax:

TimeGetSeconds()

Return:

The number of seconds the current publication has been running.

Remarks:

This function will return the number of seconds currently publication has been running.

Example:

```
var pubTimeRunning = TimeGetSeconds()
```

GetPublicationKey

Syntax:

GetPublicationKey()

Return:

A string containing the publication evaluation key.

Remarks:

This function will return the publication evaluation key. This can be used to "unlock" evaluation publications.

In order for this function to return meaningful data the Registration ID option must be enabled on the Security tab of the publication settings when the publication is created.

```
Example: var PubEvalKey = GetPublicationKey();
```

TestPublicationKey

Syntax:

TestPublicationKey(Test)

Return:

true if the given key is valid. Otherwise, false.

Parameters:

Test – the unlock code to be tested. Test must be a string or a variable containing a string. This parameter is required.

Remarks:

This function will test if a give unlock key is valid for this publication on the current computer. This can be used to "unlock" evaluation publications.

In order to operate correctly the Registration ID option must be enabled on the Security tab of the publication settings when the publication is created.

Example:

This example shows how you can implement an evaluation unlocking system. testKey is a

variable associated with a text input box. ErrorText and EnableText are text object with suitable messages in them.

The EnableText object could have an On Show trigger with a Set Publication Evaluation action to clear the evaluation flag so the user is not prompted in the future.

```
if ( TestPublicationKey( testKey ) == true ) {
      // The text object "EnableText" has an On Show trigger
      // with an action to disable the evaluation in future
      EnableText.Show() } else
{
      ErrorText.Show() }
```

Slideshow - Overview

The Slideshow functions allow you to use a script to display different slides of a Slideshow object on a page in your publication. These functions are equivalent to the Slideshow actions in the Slideshow menu of the Actions dialog.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Basic Objects

Graphical Objects

Slideshow

Functions:

Continue Continue a Slideshow

GetSlide Get the slide currently visible

GetSlideCount Count the number of slides in a slideshow

GotoSlide Go to a specific slide

IsPlaying Check if a specified Slideshow is playing

Pause Pause a Slideshow

Play Play a Slideshow

Stop Stop a Slideshow

UpdateFiles Update the "all files" list

Play

Syntax:

Play(Slide)

Parameters:

Slide - The index of the slide to begin playing from. Slide must be an integer. Slide is a zero-based index. This parameter is optional.

Remarks:

This function plays a Slideshow from the first slide if Slide is not specified.

Example:

Example 1:

For a Slideshow object named Slideshow 1, to start the slideshow from the first slide, use the following syntax:

```
Slideshow_1.Play()
```

Example 2:

To start Slideshow 1 from the third slide, use the following syntax:

```
Slideshow_1.Play(2)
```

Stop

Syntax:

Stop()

Remarks:

Call this function to stop the specified slideshow playing. The slideshow is stopped at the current slide and is not reset to the first slide in the slideshow. No parameters required.

Example:

For a Slideshow object named Slideshow 1, use the following syntax:

```
Slideshow_1.Stop()
```

Pause

Syntax:

Pause()

Remarks:

Call this function to pause the playing of the specified slideshow. No parameters required.

Example:

For a Slideshow object named Slideshow 1, use the following syntax:

```
Slideshow_1.Pause()
```

Continue

Syntax:

Continue()

Remarks:

Call this function to continue the playing of the specified slideshow. No parameters required. This function will continue from the current position in the slideshow, it will work when the slideshow has been stopped or paused.

Example:

For a Slideshow object named Slideshow 1, use the following syntax:

```
Slideshow_1.Continue()
```

IsPlaying

Syntax:

IsPlaying()

Return:

true if the slideshow is currently playing. Otherwise false.

Remarks:

This function will check if the specified slideshow is currently playing or not. No parameters required.

Example:

In this example, slideStatus will contain the return value of true or false. For a Slideshow object named Slideshow 1, use the following syntax:

```
var slideStatus = Slideshow_1.IsPlaying()
```

GotoSlide

Syntax:

GotoSlide(Slide)

Parameters:

Slide - The index position of the slide to go to. Slide must be an integer. Slide is zero-based indexed. If Slide is set to -1, the last slide is displayed.

Remarks:

This function will set the currently displayed slide in the specified slideshow to Slide. Calling this function does not make the slideshow play or stop playing, if the slideshow is currently playing it will continue to play from the specified Slide.

Example:

For a Slideshow object named Slideshow 1, to go to slide 4 in the list, use the following syntax:

```
Slideshow_1.GotoSlide(3)
```

GetSlide

Syntax:

GetSlide()

Return:

The index position of the slide that is currently visible.

Remarks:

This function will get the zero-based index position of the slide currently visible i.e. the first slide is index position 0, the second slide is index position 1, and so on. No parameters required. The slide currently displayed can be shown in a Page or Publication variable.

Example:

For a Slideshow object named Slideshow 1, to display the current slide in a Page variable named slideNum, use the following syntax:

```
slideNum = (Slideshow_1.GetSlide() + 1)
```

Note: In this example, the Page variable slideNum gets the return value of the GetSlide function and adds 1 to it. This makes the displayed value more sensible to the user as the first slide will be displayed as the number 1 and not 0.

GetSlideCount

Syntax:

GetSlideCount()

Return:

The number of slides in the specified slideshow.

Remarks:

This function returns the number of slides in the specified slideshow. No parameters required.

Example:

For a Slideshow object named Slideshow 1, use the following syntax:

```
var numSlides = Slideshow_1.GetSlideCount()
```

Note: In this example, the variable numSlides contains the number of slides in Slideshow 1. This value can be displayed in a Page or Publication variable on the current page of the publication.

UpdateFiles

Syntax:

UpdateFiles()

Remarks:

This function updates the file list in a Slideshow using the AllFiles option. No parameters required. If the slideshow is in progress it will stop and it's position will be reset.

This function is useful if the slidehow object has a varible set as the AllFiles source path. To update the files displayed by the slideshow call UpdateFiles() after changing the variable.

Example:

For a Slideshow object named Slideshow 1, use the following syntax:

```
Slideshow_1.UpdateFiles()
```

Timelines – Overview

The Timeline functions are used in a script to start or stop a Timeline object on a page in your publication. These are equivalent to the Timeline actions in the General menu of the Actions dialog.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Timelines

Basic Objects

Functions:

Start Start a Timeline

Stop Stop a Timeline

Start

Syntax:

Start()

Remarks:

This function will start a Timeline object playing. There are no parameters.

Example:

For a Timeline object named myTimeLine, use the following syntax:

```
myTimeLine.Start()
```

Stop

Syntax:

Stop()

Remarks:

This function will stop a Timeline object that is currently playing. There are no parameters.

Example:

For a Timeline object named myTimeLine, use the following syntax:

```
myTimeLine.Stop()
```

Sound - Overview

The Sound functions allow you to open a sound in a script and then play the sound in your publication. These OpusScript functions are equivalent to using the Play Sound action in the Audio/Video menu of the Actions dialog. To use these functions, you must first use the PlaySound function or the OpenSound function to create a new Sound object in the script.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Sound

Basic Objects

Functions:

GetPosition Get the current position in seconds for a sound

file

GetVolume Get the volume level of a specified device

GetVolume Get the volume level of a specified sound Object

OpenSound Open a sound file but don’t play it

Play Play a sound file

PlaySound Play a sound file

Seek Reposition the starting point of a sound

SetPosition Set the current position in seconds for a sound file

SetVolume Set the volume level for a specified device

SetVolume Set the volume level for a specified sound Object

Stop Stop a sound file

PlaySound

Syntax:

PlaySound(Sound, Preload, Times, Volume, Start, Finish, FadeIn, FadeOut, Stop, Channel, Wait)

Parameters:

Sound – The name of the sound to play. Sound should be the full pathname of the sound file or the alias name given to the sound object in the Additional Resources tab of the page or Publication Properties dialog. This parameter is required.

Note: Pathnames normally contain backslashes e.g. c:\song.wav, all single backslashes should be entered as double backslashes i.e. c:\\song.wav.

Preload – set if you want to preload a sound. Preload can be true or false. If true, the sound is preloaded. If false, the sound is not preloaded. This parameter is optional and the default is false.

Times – The number of times to play the sound. To play the sound continuously enter -1, otherwise enter a positive number. This parameter is optional. The default value is 1.

Volume – The percentage volume at which to play the sound. Volume must be a number between 0 and 100. This parameter is optional. The default value is 100.

Start – The start position, relative to the beginning of the sound, in seconds from which to play the sound. Start must be a positive number. This parameter is optional. The default is 0, which is the beginning of the sound.

Finish – The finish position, relative to the beginning of the sound, in seconds from which to finish the sound, or -1 to indicate the end of the sound. This parameter is optional. The default is -1.

FadeIn – a length in seconds in which to fade in the sound from the beginning. FadeIn must be a positive number. This parameter is optional. The default value is 0 i.e. no fade in.

FadeOut – a length in seconds in which to fade out the sound from the end. FadeOut must be a positive number. This parameter is optional. The default value is 0 i.e. no fade out.

Stop – Sets if the sound should stop when the page changes to another page. Stop can either be true or false. If Stop is true, the page will stop when the page changes. If Stop is false, the sound will continue when the page changes. This parameter is optional. The default is true.

Channel – a number indicating the mixer channel to use to play the sound. Channel must be a positive number, or the string ‘any’ to indicate that any channel can play the sound. This parameter is optional and the default is ‘any’.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function will play a specified sound with the optional parameter settings. This function can

play wave, midi, mp3 and other sound formats.

Example:

```
PlaySound("c:\\welcome.wav") // or PlaySound("intro") // intro is an alias name.
```

OpenSound

Syntax:

OpenSound(filename, channel)

Parameters:

filename – The filename of the sound to play. Filename should be the full pathname of the sound file or the alias name given to the sound object in the Additional Resources tab of the page or Publication Properties dialog. This parameter is required.

Note: Pathnames normally contain backslashes e.g. c:\song.wav, all single backslashes should be entered as double backslashes i.e. c:\\song.wav.

channel – The number of the channel on which to play the sound. Channel should be a positive number. This parameter is optional. The default is 'any' i.e. any channel can play the sound.

Return:

A Sound object set to the given filename. It can be controlled using its Play, Stop, GetPosition, SetPosition and Seek functions.

Remarks:

This function will open a sound file but not play it. This function can open wave, midi, mp3 and other sound formats.

Example:

```
var mySound = OpenSound("c:\\welcome.wav") mySound.Play()
```

Play

Syntax:

Play(Times, Volume, Start, Finish, FadeIn, FadeOut, Stop, Wait)

Parameters:

Times – The number of times to play the sound. To play the sound continuously enter -1, otherwise enter a positive number. This parameter is optional. The default value is 1.

Volume – The percentage volume at which to play the sound. Volume must be a number between 0 and 100. This parameter is optional. The default value is 100.

Start – The start position, relative to the beginning of the sound, in seconds from which to play the sound. Start must be a positive number. This parameter is optional. The default is 0, which

is the beginning of the sound.

Finish – The finish position, relative to the beginning of the sound, in seconds from which to finish the sound, or -1 to indicate the end of the sound. This parameter is optional. The default is -1.

FadeIn – a length in seconds in which to fade in the sound from the beginning. FadeIn must be a positive number. This parameter is optional. The default value is 0 i.e. no fade in.

FadeOut – a length in seconds in which to fade out the sound from the end. FadeOut must be a positive number. This parameter is optional. The default value is 0 i.e. no fade out.

Stop – Sets if the sound should stop when the page changes to another page. Stop can either be true or false. If Stop is true, the page will stop when the page changes. If Stop is false, the sound will continue when the page changes. This parameter is optional. The default is true.

Wait – Wait for this script command to complete before moving to the next script command, or not to wait. Wait is either true or false. If Wait is true, the script pauses until this animation is completed before continuing to the next line of the script. If Wait is false, the script will process the next line of script immediately. This parameter is optional. The default value is true.

Remarks:

This function will play a specified sound with the optional parameter settings. This function can play wave, midi, mp3 and other sound formats.

Example:

```
var mySound = OpenSound("c:\\intro.wav") mySound.Play()
```

Stop

Syntax:

Stop(Reset)

Parameters:

Reset – resets the sound being played to its original values. Reset can be true or false. If Reset is true, the sound is reset to its original values. If Reset is false, the sound will not be reset and can be played from the point where it was previously stopped. This parameter is required.

Remarks:

This function can stop and reset a sound file.

Example:

```
var mySound = OpenSound("c:\\intro.wav") mySound.Stop(true)
```

GetPosition

Syntax:

GetPosition()

Return:

The number of seconds for the current position for the specified Sound object.

Remarks:

This function will return the current position of the specified Sound object.

Example:

```
var mySound = OpenSound("c:\\intro.wav")
mySound.GetPosition()
```

SetPosition

Syntax:

SetPosition(From)

Parameters:

From – sets the time in seconds from which to start the specified sound. From must be a positive number. This parameter is required.

Remarks:

This function sets the current start position of the specified sound.

```
Example: var mySound = OpenSound("c:\\intro.wav") mySound.SetPosition(3)
```

Seek

Syntax:

Seek(Action, Amount)

Parameters:

Action – Action should be one of the following strings: "forward", "backward", "end", "start". This parameter is required.

Amount – Amount is the time in seconds to seek by. Amount must be a number or a variable name containing a number. This parameter is only required for the Action "forward" and "backward".

Remarks:

This function is used to re-position the starting point in the specified Sound object.

```
Example: var mySound = OpenSound("c:\\intro.wav") mySound.Seek("forward",3)
```

SetVolume

Syntax:

SetVolume(Device,Volume,Fade,Channel)

Parameters:

Device – the name of the device for which you want to change the volume. The device names are: "Wave", "Midi", "CD". The device name must be surrounded by quotes. This parameter is required.

Volume – set the volume of the device as a percentage. Volume must be a number between 0 (no volume) to 100 (full volume). This parameter is required.

Fade – the time over which the volume should fade in seconds. If no channel is set, then a value is not required.

Channel – adjust the channel the sound file is playing on. This only applies to wave files, which have been set to play on a particular channel (see PlaySound action for details on channels). The default value is -1, which is master wave volume.

Remarks:

This function will set the volume level for the specified device.

Example:

```
SetVolume("CD",100,3) // set the CD volume to 100% over 3 seconds
SetVolume("Wave",50,0,2) // set wave files playing on channel 2 to 50% volume
instantly SetVolume("Wave",25) // set master wave volume to 25% instantly
```

GetVolume

Syntax:

GetVolume(Device,Channel)

Return:

The current volume setting of the device as a percentage.

Parameters:

Device – the name of the device for which you want to get the volume. The device names are: "Wave", "Midi", "CD". The device name must be surrounded by quotes. This parameter is required.

Channel – the name of the channel you want to set. This only applies if Device is "Wave". This parameter is not required. The default value is -1 which gets the master wave volume.

Remarks:

This function will get the current volume level for the specified device.

```
Example 1: var Volume = GetVolume("CD") // get the current volume of the CD
channel
```

```
Example 2: var Volume = GetVolume("Wave") // get the master wave volume
```

Example 3:

```
var Volume = GetVolume("Wave",2) // get the wave volume of channel 2
Example 4: var Volume = GetVolume("Wave",-1) // get the master wave volume
```

SetVolume

Syntax:

SetVolume(Volume, Fade)

Parameters:

Volume – set the volume of the Sound object as a percentage. Volume must be a number between 0 (no volume) to 100 (full volume). This parameter is required.

Fade – the time over which the volume should fade in seconds. Fade must be a number. The default value is set to 0 (zero).

Remarks:

This function will set the volume level for the specified Sound object. It allows you to change the volume of a sound on a specific channel. Please note, all sounds on a particular channel will be affected by this function call. The function has been added here for your convenience.

Example:

```
var mySound = OpenSound("c:\\welcome.wav",1) // open welcome.wav on channel 1
mySound.SetVolume(0,0) // set the initial volume of the sound to zero
mySound.Play() // play welcome.wav
mySound.SetVolume(100,0.5) // fade welcome.wav in to 100% volume over half a
second
```

GetVolume

Syntax:

GetVolume()

Return:

The current volume setting of the Sound object as a percentage.

Remarks:

This function will get the current volume level for the specified Sound object.

Example:

```
var mySound = OpenSound("c:\\welcome.wav",1) // open welcome.wav on channel 1
var Volume = mysound.GetVolume() // get the current volume of the welcome.wav
and // any other sounds playing on channel 1
```

Text Objects - Overview

The Text Object functions allow you to use a script to manipulate Text objects on a page in your publication. For example, you can use the OpusScript functions listed below to create a hypertext link on a piece of text in a Text object or to play an Autonarration for a Text object.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Basic Objects

Graphical Objects

Text Objects

Functions:

CharFromPoint Get the character at a given position

CreateHypertext Create a hypertext link and action for a selection

FindText Find a string of text in a specified Text object

FindTextInSelection Find a string of text only in the

selected part of a Text object

GetFirstHypertext Get the first hypertext link in a

specified text object

GetLineCount Get the number of lines

GetNextHypertext Get the next hypertext link in a specified text object

GetNumberHypertext Get the total number of links in a specified text object

GetParagraphCount Get the number of paragraphs in a Text object

GetSelection Gets the position of the selected characters

GetSelectionParagraphStyle Get the paragraph style of selected text

GetSelectionStyle Get the formatting style of selected text

GetSelectionText Gets the selected text as a string

GetText Get a string containing the text in the hyperlink

GetTextLength Get the number of characters

GetWordCount Get the number of words

IsAutonarratePlaying Check if an autonarrate is currently playing

LineFromChar Get the line index of a character position

LineIndex Get the character index of the first character in a line

LineLength Get the number of characters in a line

ParagraphIndex Return the index number of a paragraph in a Text object

ParagraphLength Return the number of characters in a paragraph

PlayAutonarrate Begin the autonarration on this text

PointFromChar Get the position of a character

RemoveAllHypertext Remove all hypertext links for this text

ReplaceSelection Replaces the current selection with a specified string

Scroll Scroll the text by line, paragraph or page

SetColour Set the colour of the text

SetListBoxSelection Set the highlighted area in a Listbox

object to a new lin

SetSelection Selects a range of characters in the text object

SetSelectionParagraphStyle Set the paragraph style of selected text

SetSelectionStyle Set the formatting style of selected text

StopAutonarrate End the autonarration on this text

Scroll

Syntax:

Scroll(Command, Amount)

Parameters:

Command – The Command should be one of the following strings: "LineUp", "LineDown", "LineTo", "ParagraphUp", "ParagraphDown", "ParagraphTo", "PageUp", "PageDown", "PageTo", "Start", "End". This parameter is required.

Amount - Amount can be an integer or a variable name containing an integer. For "LineUp" and "LineDown" the Amount is the number of lines to move by (the default is one). For "LineTo" the Amount is the Index position of the line in the Text object i.e. the first line is index position 0, line 2 is Index position 1, and so on; the default is 0. All other Command options ignore the Amount parameter. This parameter is optional.

Remarks:

Call this function to scroll the text within a Text object using one of the Command options. This function is normally called within a Button object's Script Action or a Script Object.

Example:

For a Text object named myTextBox, use the following syntax:

```
myTextBox.Scroll("ParagraphDown") // moves down one paragraph, every time the function is called. myTextBox.Scroll("LineTo",3) // moves to the fourth line of the text box.
```

PlayAutonarrate

Syntax:

PlayAutonarrate()

Remarks:

Call this function to begin playing the autonarration of this Text object. No parameters required. This function is normally called within a Script Object when the page is triggered by an On Show trigger.

Example:

For a Text object named myTextBox, use the following syntax:

```
myTextBox.PlayAutonarrate()
```

StopAutonarrate

Syntax:

StopAutonarrate()

Remarks:

Call this function to stop playing the Autonarration for the named Text object. No parameters required.

Example:

In this example, any Autonarration currently being played for the Text object myTextBox will stop. For a Text object named myTextBox, use the following syntax:

```
myTextBox.StopAutonarrate
```

IsAutonarratePlaying

Syntax:

IsAutonarratePlaying()

Return:

true if there is an autonarration playing on the named Text object. Otherwise false. No parameters required.

Remarks:

Call this function to check if an Autonarration is currently playing for the named Text object.

Example:

In this example, the Return Value will be true if an Autonarration is currently playing for the Text object myTextBox, otherwise the Return Value will be false. For a Text object named myTextBox, use the following syntax:

```
myTextBox.IsAutonarratePlaying()
```

SetColour

Syntax:

SetColour(Colour)

Parameters:

Colour – the new colour for the text in the specified Text object. Colour must be a positive integer indicating a valid RGB value or a variable containing a valid RGB value. This parameter is optional and the default is 0 (i.e. black). Note:

The RGB value can be calculated using the RGB function. Alternatively, you can enter the red, green and blue parameters instead of the parameter Colour – see example.

Remarks:

This function will change the colour of the specified Text object to the RGB value specified in the Colour parameter.

Example:

Example 1:

To set the colour of the Text object named Intro to blue, use the following syntax:

```
NewColour = RGB(0,0,255) Intro.SetColour(NewColour)
```

Example 2:

To set the colour of the Text object named Intro by entering the red, green and blue values directly in the SetColour function, use the following syntax:

```
Intro.SetColour(255,255,0) // the text colour will be set to yellow
```

SetSelection

Syntax:

SetSelection(Start, End)

Parameters:

Start – Start should contain the index position of the character that marks the beginning of the selection. If Start is set to -1 then the selection is set to the last character's index position i.e. the selection begins at the end of the current string. If Start is set to 0 then the selection is set before the first character's index position i.e. the selection starts at the beginning of the current string. Start can be a number or a variable name containing a number. This parameter is required.

End – End should contain the index position of the character that marks the end of the selection. If End is set to -1 then the selection is set to the last character's index position i.e. the selection ends at the end of the current string. End can be a number or a variable name containing a number. This parameter is optional.

Remarks:

Selects a range of characters in the named Text object. This function is necessary when you want to modify text within a Text object or query its current value. When Start equals End the selection is set for insertion. For a Text Input object, the selection will be visible as an inverted region, or a flashing cursor.

Example 1:

For a Text Input object named myTextInput, use the following syntax:

```
myTextInput.SetSelection(-1) // sets the insertion point at the end of the
text in myTextInput myTextInput.SetSelection(0,-1) // Selects the whole of
the text in myTextInput myTextInput.SetSelection(5,5) // Sets the insertion
point at the fifth character in myTextInput
```

Example 2:

This function is normally used with the ReplaceSelection function. For example, if myTextInput contains "Mr Smith", the following syntax will insert the first name ‘James’ in myTextInput

```
myTextInput.SetSelection(3,3) myTextInput.ReplaceSelection("James ") Note:
```

In this example, myTextInput now contains "Mr James Smith".

GetSelection

Syntax:

GetSelection()

Return:

This Return Value is a new object with two properties: start and end.

start – start is an integer showing the Index position of the first character of the selection. The first character of a string is Index position 0, the second character is Index position 1, and so on i.e. the first character displayed in a text box has an Index position of 0.

end - end is an integer showing the Index position of the last character of the selection.

Remarks:

This function returns the Index position of the first and last characters currently selected in a Text object. No parameters are required. This function is normally used in conjunction with the SetSelectionfunction.

Example:

For a Text object named myText containing the value Mr James Smith, use the following syntax:

```
var selectedArea = myText.GetSelection() Note:
```

If the currently selected text within the Text object myText was the word James, the Return Value of start would be index position 3, and for end, index position 8. In this example, the selectedArea variable becomes an object that has two properties named start and end. To check the value of start or end save the property to a variable in a Script or as a Page or

Publication variable displayed on a page within Opus.

For example, to save the properties to two Script variables named startSelection and endSelection

```
var startSelection = selectedArea.start var endSelection = selectedArea.end
```

GetSelectionText

Syntax:

GetSelectionText()

Return:

The text currently selected as a string.

Remarks:

This function gets the selected text as a string. No parameters required. This function is normally used with the SetSelectionand ReplaceSelectionfunctions.

Example:

To extract the first name from a Text object named fullName containing the value Mr James Smith, use the following syntax:

```
fullName.SetSelection(3,8) myFirstName = firstName.GetSelectionText()
firstName.ReplaceSelection(myFirstName) Note:
```

In this example, the last line of code adds the Return Value James from the GetSelectionText function to a Text object named firstName. The Text object now contains the text James.

ReplaceSelection

Syntax:

ReplaceSelection(String)

Parameters:

String - a string to replace the current selection within a Text object. String can be any string surrounded by quote marks or a variable name containing a string.

Remarks:

This function replaces the currently selected text in a Text object with String. If no text is currently selected in the Text object, then String is inserted at the start of the current text. This function is normally used after a SetSelection function has selected a range of characters within a text field.

Example:

To replace the complete contents of a Text object named myText with a new string, use the following syntax:

```
myText.SetSelection(0,-1) myText.ReplaceSelection("This is my new text")
```

FindText

Syntax:

FindText(String, StartPosition)

Return:

The Index position of the first character in the parameter String if it is found and the value -1 if it is not found.

Parameters:

String - a string to find within a Text object. String can be any string surrounded by quote marks or a variable name containing a string. This pararemeter is required.

StartPosition – the Index position to start the search at. StartPosition is a zero-based index. This parameter is optional and the default is to start at index number 0.

Remarks:

This function allows you to find a character, word or phrase within a Text object. You can start the search from the beginning of the text or at any point within the text. If used with a while loop you can find the number of occurrences of the string within the text.

Example 1:

To find the phrase .co.uk in a Text object named Emails starting from the beginning of the text, use the following syntax:

```
Emails.FindText(".co.uk",0)
```

Example 2:

To find the number of occurrences of the phrase .co.uk in a Text object named Emails starting from the beginning of the text, use the following syntax:

```
numOccur = 0 Counter = 0 indexPos = 0 while (numOccur != -1) {
numOccur = Emails.FindText(".co.uk",indexPos) indexPos = numOccur + 1
Counter++ } Counter -= 1 Note:
```

In the example above, the variable Counter will contain the number of occurrences of the phrase .co.uk. Because the while loop will always run through one extra time, you need to deduct 1 from the end result i.e. the last line of code Counter = 1.

FindTextInSelection

Syntax:

FindTextInSelection(String, StartPosition)

Return:

The Index position of the first character in the parameter String if it is found and the value -1 if it is not found.

Parameters:

String - a string to find within a Text object. String can be any string surrounded by quote marks or a variable name containing a string. This pararemeter is required.

StartPosition – the Index position to start the search at. StartPosition is a zero-based index. This parameter is optional and the default is to start at index number 0.

Remarks:

This function allows you to find a character, word or phrase within a Text object. This function will only search for the parameter String in the selected text of a Text object, the rest of the text is ignored – see SetSelection for more information on selecting text. If used with a while loop you can find the number of occurrences of the string within the text. Note:

If no text is selected, this function will search through all of the text in the specified Text object.

Example:

To find the phrase .co.uk in a Text object named Emails starting from the beginning of the text to the index position 15, use the following syntax:

```
Emails.SetSelection(0,15) foundText = Emails.FindTextInSelection(".co.uk",0)
if (foundText == -1) {
      Debug.trace("No occurrences of '.co.uk' found!") } else {
      // other statements here }
```

Note:

In the example above, the if statement will only run when the phrase .co.uk is not found. If the phrase is found, the lines of code in the else section will be run.

GetParagraphCount

Syntax:

GetParagraphCount()

Return:

An integer indicating the number of paragraphs in the specifed Text object.

Remarks:

This function returns the number of paragraphs in a specifed Text object. The number is equal to the number of carriage returns entered in the text box. A carriage return is inserted every time you hit the Enter/Return key on your key board. Therefore, if you have two paragraphs with a single blank line separating them, the number returned with this function will be 3 because a carriage return is inserted on the blank line.

Example:

To find the number of paragraphs in a Text object named IntroText and to store the return value in a variable named NumParas, use the following syntax:

```
NumParas = IntroText.GetParagraphCount()
```

ParagraphIndex

Syntax:

ParagraphIndex(ParaIndex)

Return:

An integer indicating the Index position before the first character in the paragraph specifed in the parameter ParaIndex. If the number entered in ParaIndex is not a valid paragraph, the Return Value is -1.

Parameter:

ParaIndex – the number of the paragraph you want to check. ParaIndex must be an integer. The paragraphs are numbered using a zero-based index. This parameter is required and the default is 0 (i.e. the first paragraph).

Remarks:

This function returns the Index position before the first character of the specified paragraph in a Text object. This function is often used with the ParagraphLength function and is useful if you want to replace, insert or select text in a particular paragraph.

Example:

In the example below, the variable ParaNum will contain the Index number for the third paragraph of the Text object Text1. The SetSelection function uses the number in ParaNum as the starting point for a selection – because this function does not specify an end point, the ReplaceSelection function on the next line will insert the string "Digital Workshop" at the beginning of the third paragraph of the Text object named Text1. Any other text in that paragraph will appear to the right of the inserted text.

```
ParaNum = Text1.ParagraphIndex(2) Text1.SetSelection(ParaNum)
Text1.ReplaceSelection("Digital Workshop")
```

ParagraphLength

Syntax:

ParagraphLength(ParaIndex)

Return:

An integer indicating the number of characters in the paragraph specified in the parameter ParaIndex. If the number entered in ParaIndex is not a valid paragraph, the Return Value is -1.

Parameter:

ParaIndex – the number of the paragraph you want to check. ParaIndex must be an integer. The paragraphs are numbered using a zero-based index. This parameter is required and the default is 0 (i.e. the first paragraph).

Remarks:

This function returns the number of characters in a paragraph. This function is often used with the ParagraphIndex function and is useful if you want to replace, insert or select text in a particular paragraph

Example:

In the example below, the variable ParaNum and ParaLengthwill contain the Index number for the third paragraph of the Text object Text1. The SetSelection function uses the number in ParaNum as the starting point for a selection and ParaLength for the end point of a selection. The ReplaceSelection function on the next line will replace the current contents of the third paragraph with the string "Digital Workshop" in the Text object named Text1.

```
ParaNum = Text1.ParagraphIndex(2) ParaLength = Text1.ParagraphLength(2)
Text1.SetSelection(ParaNum,ParaLength) Text1.ReplaceSelection("Digital
Workshop")
```

GetSelectionStyle

Syntax:

GetSelectionStyle()

Return:

A new object with some of the following properties. A property is only present within the object if it is consistently used across the whole selection. For example, if a word in bold is selected the bold property is true, if a word in plain text is selected the bold property is false, and if two words, one bold and the other plain, are selected then the bold property is not present. The object properties are:

bold - true if the text is bold. Otherwise false.

italic - true if the text is italic. Otherwise false.

underline - true if the text is underlined. Otherwise false.

subscript - true if the text is in subscript. Otherwise false.

superscript - true if the text is in superscript. Otherwise false.

fontname - The name of the font face.

fontsize - The size of the font in points.

fontaspect – The aspect of the font in integers.

colour - The colour of the text in RGB values as a string e.g. "00FF00".

backgroundcolour - The colour of the background of the text in RGB values as a string, or the string "transparent".

shadowcolour - The colour of the shadow of the text in RGB values as a string, or the string "none".

Remarks:

No parameters required for this function. A new object that contains the properties outlined above. This function can be used to determine the style of one Text object that can then be applied to a different Text object using the SetSelectionStyle function so that both Text objects have the same format.

Example:

To find the current font name used for a Text object named myText, use the following syntax:

```
myText.SetSelection(0,-1) var myTextFormat = myText.GetSelectionStyle() var
myTextFont = myTextFormat.fontname Note:
```

You must first set the selection of the Text object or the Return Values for this function will be 'undefined'. In this example, the whole Text object myText was selected and the Return Values for GetSelectionStyle saved in a variable named myTextFormat. The variable myTextFormat is now an object that contains the Return Values as properties. In this example, to save the property fontsize in a new variable named myTextFont the fontsize property is identified in the myTextFormat object i.e. myTextFormat.fontsize. To identify the bold Return Value use, myTextFormat.bold; to identify the italic Return Value, use myTextFormat.italic, and so on.

SetSelectionStyle

Syntax:

SetSelectionStyle(Style)

Parameters:

Style – Style is a reference to one property of an object that can be used to set one of the style formats of a Text object. For example, Style could make the text within a Text object bold, italic, underline, subscript or superscript. Or Style could change a Text objects, font, fontsize, fontaspect, colour, backgroundcolour or shadowcolour. For correct syntax, refer to the GetSelectionStyle function. This parameter is required.

Remarks:

This function allows you to set a new formatting style to a Text object. Only the properties specified are applied to the text.

Example:

For example, to set the Text object named myText to all bold use the following syntax:

```
myText.SetSelection(0,-1) var newStyle = new Object() newStyle.bold = true
myText.SetSelectionStyle(newStyle)
```

GetSelectionParagraphStyle

Syntax:

GetSelectionParagraphStyle()

Return:

A new object with some of the following properties. A property is only present within the object if it is consistently used across the whole selection. For example, if the selection includes more than one paragraph and the justification is different for each paragraph, the justification property will not appear in the new object. To ensure this does not happen, you should ensure you only select one paragraph at a time with this function. The object properties are:

justification – this is a string showing the selections justification. The string will contain one of the following key words: left, right, centre, full, density or aspect.

linespacing – this is a string showing the method of spacing for the paragraph. The string will begin with one of the following key words: x, = or At Least.

Note:

(i) If it begins with x (e.g. x1, x2) it is a multiple. For example x2 means the linespacing is double-spaced, x3 triple-spaced, and so on.

(ii) If the string begins with = (e.g. =12pt, =20pt) it is an exact height. This means, the spacing is a specific size and the line is always exactly this high.

(iii) If the string is At Least the line height is a specific size. This means, 'at least this high', therefore, larger text will force the line to be higher.

spacingbefore – this is a number indicating the point size of the spacing between the last paragraph and the current paragraph.

spacingafter – this is a number indicating the point size of the spacing after the current paragraph and the next paragraph.

leftindent – this is a number indicating the number of pixels spaces from the left edge of the Text object and the first letter on a line in the paragraph.

rightindent – this is a number indicating the number of pixel spaces from the right edge of the Text object and the last letter on a line in the paragraph.

firstlineindent – this is a number indicating the number of pixel spaces from the left edge of the Text object and the first letter on the first line of text in a paragraph.

hangingindent – this is a number indicating the number of pixel spaces have been set for a hanging indent in a Text object.

bullet – a string indicating the type of bullet used for the paragraph. The string will contain one of the following key words: no bullet, character or graphical.

Remarks:

This function will return a new object containing information about the paragraph's style settings, such as, if the paragraph is left or right justified, uses single spacing or hanging indents, and so on. This function will only work if you currently have text in a Text object selected using the SetSelection function. If more than one paragraph is selected and they have different settings

the new object will not contain that property.

This function can be used to determine the style of one Text object in your publication that you can then apply to a different Text object using the SetSelectionParagraphStyle function. Alternatively, the SetSelectionParagraphStyle function can be used to change one or more of the paragraph styles of the Text object specified with this function to a new style.

Example 1:

To get the current paragraph style settings for the second paragraph in a Text object called IntroText, use the following syntax:

```
ParaNum = IntroText.ParagraphIndex(0) ParaLength =
IntroText.ParagraphLength(0) IntroText.SetSelection(ParaNum,ParaLength) var
paraInfo = IntroText.GetSelectionParagraphStyle() Note:
```

The new object named paraInfo will contain all of the paragraph style information. For example, if you wanted to view the current style of justification and line spacing in the paragraph, use the following syntax:

```
Debug.trace(paraInfo.justification + "\n") Debug.trace(paraInfo.linespacing)
Note:
```

The + "\n" part of the first line of code means to add a new line in the Debug window that appears when the Debug.trace function is run – this is purely to ensure the two bits of information appear on separate lines.

Example 2:

In the example below, the script is extended to show how you could change the justification for the second paragaph of the Text object named IntroText if it is currently not set to centre justification.

```
ParaNum = IntroText.ParagraphIndex(1) ParaLength =
IntroText.ParagraphLength(1) IntroText.SetSelection(ParaNum,ParaLength) var
paraInfo = IntroText.GetSelectionParagraphStyle()

if (paraInfo.justification == "Left") {

paraInfo.justification = "Centre"

IntroText.SetSelectionParagraphStyle(paraInfo) }
```

SetSelectionParagraphStyle

Syntax:

SetSelectionParagraphStyle(Style)

Parameters:

Style – Style is a reference to one property of an object that can be used to set one of the paragraph style formats for a Text object. For example, Style could change the following paragraph styles within a Text object: justification, linespacing, spacebefore, spaceafter, leftindent, rightindent, firstlineindent, hangingindent or bullet. For correct syntax, refer to the

GetSelectionParagraphStyle function. This parameter is required.

Remarks:

This function allows you to set a new paragraph style to a Text object. Only the properties specified are applied to the text. This function will only work if you have a paragraph selected in a Text object, use the SetSelection function to create a selection.

Example 1:

In this example, the second paragraph of a Text object named IntroText is set to have a left justification and a bullet point. A new object named newStyle is created that contains the new styles you want to set for IntroText

```
ParaNum = IntroText.ParagraphIndex(0) ParaLength =
IntroText.ParagraphLength(0) IntroText.SetSelection(ParaNum,ParaLength) var
newStyle = new Object() newStyle.justification = "Centre"
newStyle.bullet.type = "Character" newStyle.bullet.character = "a"
newStyle.bullet.font = "WingDings"
newStyle.bullet.colour = RGB(255,128,64)
IntroText.SetSelectionParagraphStyle(newStyle)
```

Example 2:

In this example, the second paragraph of a Text object named MoreInfo is set to the same paragraph style as the first paragraph in the Text object named IntroText

```
ParaNum = IntroText.ParagraphIndex(0) ParaLength =
IntroText.ParagraphLength(0) IntroText.SetSelection(ParaNum,ParaLength) var
paraInfo = IntroText.GetSelectionParagraphStyle() ParaNum =
MoreInfo.ParagraphIndex(1) ParaLength = MoreInfo.ParagraphLength(1)
MoreInfo.SetSelection(ParaNum,ParaLength)
MoreInfo.SetSelectionParagraphStyle(paraInfo) Note:
```

In Example 1, a new object was created in OpusScript and the style you required added to it. In Example 2, an existing Text object on the current page of a publication was used to copy a paragraph style and apply it to another Text object.

SetListBoxSelection

Syntax:

SetListBoxSelection(Index)

Parameters:

Index – the Index position of the line you want to select. Index is a zero-based index. If Index is set to –1 then the current list box selection is cleared. This parameter is required.

Remarks:

This function allows you to highlight a particular line in a Listbox object. It can also be used to clear the currently highlighted area of a list box – it does not delete the line's text.

Note:

This function is for Listbox objects only. Using this function with other Text objects will have no effect.

Example 1:

To select the first item in a Listbox object named Listbox 1, use the following syntax:

```
Listbox_1.SetListBoxSelection(0)
```

Example 2:

To clear the current selection for a Listbox object named Listbox 1, use the following syntax:

```
Listbox_1.SetListBoxSelection(-1)
```

PointFromChar

Syntax:

PointFromChar(CharIndex)

Return:

This function Returns an object containing the coordinates of the specified character as these properties:

x - the x coordinate of the top-left of the character relative to the text object.

y - the y coordinate of the top-left of the character relative to the text object.

width - the width of the character.

height - the height of the character.

Parameters:

CharIndex – The Index position of the character within the specified Text object. The index is a zero-based index. CharIndex must be an integer or a variable name containing an integer. This parameter is required.

Remarks:

The Return Values describe the position for the text without any effects or animations applied (rotation/scale etc.).

Example:

For a Text object named myText, use the following syntax:

```
var charInfo = myText.PointFromChar(1) Note:
```

In this example, the new variable charInfo becomes an object with four properties. To reference one of the properties use the following syntax: charInfor.x; charInfo.y; charInfo.width; charInfo.height.

CharFromPoint

Syntax:

CharFromPoint(PosX, PosY)

Return:

The index position of the character closest to the given position.

Parameters:

PosX - The x coordinate of the top-left of the character relative to the Text object.

PosY - the y coordinate of the top-left of the character relative to the Text object.

Remarks:

The position values describe the position assuming no effects or animations applied (rotation/scale etc.).

Example:

For a Text object named myText, use the following syntax:

```
var charInfo = myText.CharFromPoint(37,23)
```

LineFromChar

Syntax:

LineFromChar(CharIndex)

Return:

The index position of the line on which the CharIndex was found.

Parameters:

CharIndex - The Index position of the specified character, the index is a zero-based index. CharIndex must be an integer. This parameter is required.

Remarks:

This function will return the line number on which the CharIndex was found. The lines within a Text object also start with a zero index position.

Example:

For a Text object named myText containing four lines each of which contains two characters per line, use the following syntax to return the line number for the fifth character in myText

```
var lineInfo = myText.LineFromChar(5) Note:
```

In this example, the Return Value contained in the variable lineInfo is 1, which is the second line of the Text object myText.

LineIndex

Syntax:

LineIndex(LineIndex)

Return:

The index position of the first character of the line specified in LineIndex.

Parameters:

LineIndex - The index position of the line. Lines are zero-based indexed. LineIndex must be an integer. This parameter is required.

Remarks:

This function will return the index position of the first character of the line specified in LineIndex. The character index position is also zero-based indexed i.e. the first character in the Text object is index position 0, the second character index position 1, and so on.

Example:

For a Text object named myText, to find the index position of the first character on the third line, use the following syntax:

```
var charInfo = myText.LineIndex(2)
```

LineLength

Syntax:

LineLength(LineIndex)

Return:

The number of characters in the line specified in LineIndex.

Parameters:

LineIndex - The index position of the line. Lines are zero-based indexed. LineIndex must be an integer. This parameter is required.

Remarks:

This function will return the number of characters contained on a specified line. If the line specified in LineIndex contained 5 characters, the Return Value will be 6, if the line had 13 characters, the Return Value will be 14.

Example:

For a Text object named myText, to find the number of characters on the first line of text, use the following syntax:

```
var numOfChars = myText.LineLength(0)
```

GetLineCount

Syntax:

GetLineCount()

Return:

The number of lines in the specified Text object.

Remarks:

No parameters required.

Example:

For a Text object named myText, use the following syntax:

```
var numOfLines = myText.GetLineCount()
```

GetWordCount

Syntax:

GetWordCount()

Return:

The number of words in the specified Text object.

Remarks:

No parameters required.

Example:

For a Text object named myText, use the following syntax:

```
var numOfWords = myText.GetWordCount()
```

GetTextLength

Syntax:

GetTextLength()

Return:

The number of characters in the specified Text object.

Remarks:

No parameters required.

Example:

For a Text object named myText, use the following syntax:

```
var numLength = myText.GetTextLength()
```

CreateHypertext

Syntax:

CreateHypertext(Action)

Parameters:

Action - A function or string to be called to perform the hypertext's Left-click action. This parameter is required.

Remarks:

This function is used to create a hypertext link and perform the specified Action whenever the mouse is left-clicked over the hypertext area. More than one hypertext link can be created in the same Text object, however each new hypertext link will require a new CreateHypertext function.

Example:

For a Text object named myText containing the text 'Click here or there' where the word 'here' would display an Image object named Image1 and the word 'there' would display an Image object Image2, use the following syntax in myText (note: write the Script Action with the trigger On Show)

```
myText.SetSelection(6,10) myText.CreateHypertext(showImage())

myText.SetSelection(14,19) myText.CreateHypertext("Image2.Show()") Note:
```

In this example, the Action in the first CreateHypertext function calls a function named showImage. This function must either be written in a Script Object on the current page. The Action in the second CreateHypertext function is a string containing the expression to show Image2. Neither the function or script will run until the user left-clicks on the hypertext area.

GetFirstHypertext

Syntax:

GetFirstHypertext()

Return:

The hyperlink or a null value if there is no hyperlink or it is already the first hyperlink in the specified Text object. The hyperlink is a Hypertext object that can be used with the GetNextHypertext function or the GetText function.

Remarks:

This function gets the first hypertext link in a Text object.

Note:

All hyperlinks in a Text object are returned in the reverse order to that shown in the text box. In other words, the last hyperlink in the text box is the first hyperlink retrieved in the Return Value.

Example:

For a Text object named Text 1, use the following syntax:

```
var Link = Text_1.GetFirstHypertext()
```

Note:

The Return Value will be a new Hypertext object in the variable named Link. The Hypertext object is used as the parameter in the GetNextHypertextfunction.

Use the GetText function if you want to view the hyperlink's text (i.e. the underlined hypertext in the Text object).

GetNextHypertext

Syntax:

GetNextHypertext(Link)

Return:

The hyperlink or a null value if there is no hyperlink in the specified Text object.

Parameter:

Link – the name of a Hypertext object returned from a previous GetFirstHypertext or GetNextHypertext statement.

Remarks:

This function gets the next hypertext link in a Text object. You must have used the GetFirstHypertextfunction before you can use this function.

Note:

All hyperlinks in a Text object are returned in the reverse order to that shown in the text box. In other words, the last hyperlink in the text box is the first hyperlink retrieved in the Return Value.

Example:

For a Text object named Text 1, use the following syntax:

```
var Link = Text_1.GetFirstHypertext() Debug.trace(Link.GetText() + "\n") var
nextLink = Text_1.GetNextHypertext(Link) Debug.trace(nextLink.GetText() +
"\n") Note:
```

The GetText function displays the hyperlink's text – in this example, the text is displayed in a Debug window. If the GetFirstHypertext function has not previously been used for the Text object named Text 1, the GetNextHypertext link would not work.

To show the text for each hyperlink in a Text object, use the for loop with the GetNumberHypertext function – see GetNumberHypertext example for more information.

GetText

Syntax:

GetText()

Return:

A string containing the text in the hyperlink.

Remarks:

This function gets the actual text contained in the hyperlink. Note:

This function can only be used with the hypertext functions.

Example:

To display the text used for the hyperlink in a Text object named Intro in a Debug window, use the following syntax:

```
var Link = Intro.GetFirstHypertext() Debug.trace(Link.GetText())
```

GetNumberHypertext

Syntax:

GetNumberHypertext()

Return:

The number of hyperlinks in a specified Text object.

Remarks:

This function gets the total number of hypertext links in a Text object.

Example 1:

For a Text object named Text 1, use the following syntax:

```
var numLinks = Text_1.GetNumberHypertext()
```

Example 2:

To display the text for all hyperlinks in a Text object named Text 1, use the following syntax:

```
var numLinks = Text_1.GetNumberHypertext() if (numLinks !=0) {
link = Text_1.GetFirstHypertext() Debug.trace(link.GetText() + "\n") nextLink
= link for (loop=1;loop<=numLinks-1;loop++) {
                  newLink = Text_1.GetNextHypertext(nextLink)
Debug.trace(newLink.GetText() + "\n") nextLink = newLink
      } }
```

Note:

In Example 2 above, the variable numLinks contains the number of hyperlinks in the Text object named Text_1. The lines of code in the If statement are run when numLinks is not zero (i.e. it contains at least one hyperlink). The variable link will contain the first hyperlink, which is then

displayed in a Debug window using the Debug.trace function. The variable nextLink is used as the parameter for the GetNextHypertext function. The first time it is used, nextLink is set to the first hyperlink (i.e. nextLink = link) and then it is set in the for loop to the next hyperlink (i.e. nextLink = newLink).

The for loop starts at a count of 1 because the first hypertext has already been displayed. The loop will continue for the total number of hyperlinks minus 1 (because we have already displayed the first hypertext), so we need to make sure we loop 1 less than the total number of hyperlinks in the Text object.

RemoveAllHypertext

Syntax:

RemoveAllHypertext()

Remarks:

No parameters required. This function will remove all hyperlinks from the specified Text object.

Example:

For a Text object named myText, to remove all hypertext links currently set within the Text object, use the following syntax:

```
myText.RemoveAllHypertext()
```

Video - Overview

The Video functions allow you to use a script to control a Video object on a page in your publication. For example, you can use the Video functions to check if a video is playing or to start and stop a video from playing, as well as go to specific parts of the video or find out how long the video has been playing for.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Basic Objects

Graphical Objects

Video

Functions:

GetLength Get the length of the video in seconds

GetPosition Get the current position of a video in seconds

Go Set the first frame of a video

IsPlaying Check if a specified video is playing

Play Play a specified video

Seek Reposition the starting point of a video

Stop Stop a specified video

Play

Syntax:

Play(From, To, Synch)

Parameters:

From - A position in seconds to play from. From must be a number or a variable name containing a number (e.g. 1, 2, 3 etc. or 0.2, 0.4, 0.6 etc are valid entries). This parameter is optional.

To - A position in seconds to play to. To must be a number or a variable name containing a number. A value of -1 indicates play to the end. This parameter is optional.

Synch – Synch can be set to true or false. If Synch is true, synchronise message is sent. If Synch is false, synchronise message is NOT sent. This parameter is optional.

Remarks:

This function will play the video in a Video object. If the From and To parameters are not used, this function plays the whole video.

Example:

For a Video object named myVideo, use the following syntax:

```
myVideo.Play() // or myVideo.Play(2,-1) // or myVideo.Play(2,-1,false) //
synch message not sent.
```

Stop

Syntax:

Stop(Reset, Synch)

Parameters:

Reset – Reset the video to the start. If Reset is true, the video is reset to the start. If Reset is false, the video is not reset and remains at its current position. This parameter is optional. If Reset is not entered, the default value is false.

Synch – To set the synchronise to end message. If Synch is true, the synch message is sent. If Synch is false, the synch message is not sent. This parameter is optional. If Synch is not entered, the default value is true.

Remarks:

This function will stop the video playing in a Video object. Optional parameters allow the video to be reset to the start of the video or set to send the sync message.

Example:

For a Video object named myVideo, use the following syntax:

```
myVideo.Stop() // or myVideo.Stop(true, true) // resets the video to start
and sends synch message. // or myVideo.Stop(false, false) // stops video at
```

```
current position and does not send synch message.
```

Go

Syntax:

Go(Position)

Parameters:

Position - A position in seconds to play from. Position must be a number or a variable containing a number. If Position is set to -1, the video is set to the end frame. This parameter is required.

Remarks:

This function will set the first frame of the video to Position. This function is only required when you want to play a video from a different starting point than the beginning of the video. This function is normally set before the Play function.

Example:

For a Video object named myVideo, use the following syntax:

```
myVideo.Go(3) myVideo.Play()
```

IsPlaying

Syntax:

IsPlaying()

Return:

true if the video in the specified Video object is currently playing. Otherwise false.

Remarks:

This function will check if the video in the specified Video object is currently playing or not. No parameters are required.

Example:

For a Video object named myVideo, use the following syntax:

```
var vidStatus = myVideo.IsPlaying() Note:
```

In this example, the variable vidStatus will contain either the Return Value true or false.

GetPosition

Syntax:

GetPosition()

Return:

The current position of the specified Video object in seconds.

Remarks:

This function returns the current time the video has been playing. No parameters required. Note:

There is also a GetPosition function in the Graphical Objects folder that returns a new object containing the x and y coordinates of the graphical object you specified. To get the x and y coordinates of a Video object, use the GetXPosition and GetYPositionfunctions.

Example:

For a Video object named myVideo, use the following syntax:

```
var videoPos = myVideo.GetPosition() Note:
```

In this example, the variable videoPos will contain the Return Value of the current position of the video myVideo in seconds.

GetLength

Syntax:

GetLength()

Return:

The length of the specified Video object in seconds.

Remarks:

No parameters required.

Example:

For a Video object named myVideo, use the following syntax:

```
var videoLength = myVideo.GetLength() Note:
```

In this example, the variable videoLength will contain the Return Value of the total length of the video myVideo in seconds.

Seek

Syntax:

Seek(Action, Amount)

Parameters:

Action – Action should be one of the following strings: "forward", "backward", "end", "start". This parameter is required.

Amount – Amount is the time in seconds to seek by. Amount must be a number or a variable name containing a number. This parameter is only required for the Action "forward" and "backward".

Remarks:

This function is used to re-position the starting point in the specified Video object.

Example:

For a Video object named myVideo, use the following syntax:

```
myVideo.Seek("forward",3.479) // or myVideo.Seek("end")
```

Views - Overview

The View functions allow you to use a script to control windows in your publication. The Main Publication Window and other popup windows you can create in your publication are known in OpusScript as Views. You can use the View functions for example, to check how many windows are currently open in your publication or to close specific windows down.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Views

Basic Objects

Functions:

Close Close the view

GetFirstView Get the first view open

GetNextView Get the next view

GetNumberViews Get the number of views open

GetPage Get the page currently displayed in the view

GetType Get the View Type

GetViewAt Get the view at a specified index position

GetFirstView

Syntax:

GetFirstView()

Return:

The first view currently open.

Remarks:

This function will get the first view currently open. While the publication is running, Opus maintains a list of all the views that are open. The GetFirstView and GetNextView functions allow access to the list. However, the list is not guaranteed to be in any particular order. No parameters required.

```
Example: var view = GetFirstView()
```

GetViewAt

Syntax:

GetViewAt(Index)

Return:

The view at the specified index position.

Parameters:

Index – the Index number of a view. This parameter is required.

Remarks:

This function will get the view at the specified index position. This function is used with the GetNumberViewsto iterate all the views.

```
Example: var view = GetViewAt(1)
```

GetNumberViews

Syntax:

GetNumberViews()

Return:

The number of views currently open in the publication.

Remarks:

This function will return the number of views (Opus windows) currently open in the publication. No parameters required.

Example:

```
var numViews = GetNumberViews()
```

GetPage

Syntax:

GetPage()

Return:

The Page object currently displayed in the view.

```
Example: var view = GetFirstView() view.GetPage()
```

Close

Syntax:

Close()

Remarks:

Closes the view. Trying to close a view that is not in an external window will do nothing.

```
Example: var view = GetFirstView()
view.Close()
```

GetType

Syntax:

GetType()

Return:

The type of panel the view is. The possible return values are:

Value Meaning

0 Main Window

1 Panel 1

2 Panel 2

3 Panel 3

-1 Free floating window

-2 An error occurred

Remarks:

This function will return the type of view for the specified Object.

```
Example: var view = GetFirstView() var type = view.GetType()
```

Database Functions - Overview

Opus Pro provides the ability to connect to a database. Please note that it is intended to provide an easy route to display the contents of individual records and not huge lists of records. Nor should it be seen as a replacement for dedicated tools for more sophisticated database programming requirements.

Note:

A full description of databases is provided in the main Opus Help file. To open the help, click on Contents & Index option in the Help menu at the top of the Opus Editor. Search for the word databases in the Index tab of the help file for more information.

The Database OpusScript functions are unique to Database objects created in a script. In other words, you must create a new Database object in your script that references a DSN, before you can use the OpusScript Database functions.

Below are three , showing you how you can create a new Database object:

For a file DSN on drive d:

```
var myDB = new Database("FILEDSN=d:\catalogue.dsn;")
```

For a DSN in default path of Opus:

```
var myDB = new Database("FILEDSN=catalogue.dsn;")
```

```
For Machine DSN's: var myDB = new Database("DSN=catalogue;")
```

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Database objects:

Class Hierarchy:

Database

Functions:

AutoCommit Turn AutoCommit on or off for a database

Commit Commit to database – permanently

stored on database

Connect Connect to SQL server

ExecuteSQL The SQL database command to execute

FirstRecord Go to First record in record set

GetCurrentRecordNumber Get current record number in record set

GetIdentQuoteChar Identify quote character required for

table or field names containing spaces

GetLastError Get the last database error message

GetNumberOfRecords Get total number of records in recordset

GetRecordAt Go to a specific record number in arecordset

GetRecordAtRelative Go to a specific record number relative to current number

IsAutoCommitOn Check if AutoCommit is on in a database

LastRecord Go to Last record in record set

NextRecord Go to Next record in record set

PreviousRecord Go to Previous record in record set

Rollback Go back to point in database before changes were made or last commit sent

Database Functions - Introduction

Below is a simple OpusScript example showing how you can use the OpusScript Database functions in a Script Object or Script Action. The example shows in which order you should use the Database functions.

The full script is:

```
var db = new Database("DSN=Products;") var RecSet = db.ExecuteSQL("SELECT Price, Qty FROM Customer_Order WHERE OrderID=1;") var total = 0

// Setup loop for all the records found for (var i=1; i<= RecSet.GetNumberOfRecords(); i++) {

total += RecSet.Price * RecSet.Qty RecSet.NextRecord() // Move to the next record }
```

Explaining the Example:

1. The first thing you must do when using the OpusScript Database

functions is to create a new Database object in your script:

```
var db = new Database("DSN=Products;")
```

Note:

In this example, the new Database object is connecting to a database named Products, which contains the data you want to work with. It connects to the database using the Data Source Name (DSN) protocol, which is described in Connecting to a Database in the main Help file.

2. Once you have established a connection to a database, your next task is to get the script to work with the exact data you want from the database. A database simply stores information in a structured format. Typically, a database is made up of a number of Tables that contain records. Each record contains fields that hold information. In this example, the database is called Products and it contains a Table named Customer_Order, every record in this table contains a Price, Qty and OrderID field.

Note:

A language called SQL (Structured Query Language) is typically used to select the exact data you want from a database. Any records that match the SQL statement are saved in a Record Set.

In OpusScript, the ExecuteSQL function allows you to enter an SQL statement, for example

```
 var RecSet = db.ExecuteSQL("SELECT Price, Qty FROM Customer_Order WHERE OrderID=1;")
```

In this example, the ExecuteSQL function will select the Price and Qty fields from the table Customer_Order if the OrderID field contains the value 1. Each record that matches this SQL statement will be saved into the variable RecSet – this is the Record Set for this SQL statement.

Certain OpusScript Database functions work only with the Record Set that is created, these include the following functions: NextRecord, PreviousRecord; FirstRecord, LastRecord, GetRecordAt, GetRecordAtRelative, GetNumberOfRecords and GetCurrentRecordNumber. In

order to use these functions, you must refer to the variable that contains the name of the Record Set (in this example, the variable is named RecSet). Three are given in this example and are described in the next paragraphs.

3. Now that you have built a Record Set of records that match your selection criteria, you can use the other OpusScript Database functions to perform a variety of other useful operations. In this example, a for loop function is used with the Database function GetNumberOfRecords:

```
for (var I=1; I <= RecSet.GetNumberOfRecords(); I++)
```

Note: The GetNumberOfRecords function will find the number of records contained in the variable RecSet (the new Record Set this script has just created). It uses the number of records in the Record Set to calculate the number of times it must run the commands contained inside the loop.

4. The actual loop contains two commands. The first command is

this:

```
total += RecSet.Price * RecSet.Qty
```

This command performs a calculation and adds the result of the calculation to the variable called total. The calculation uses the current value contained in the Price field of the currently selected record in the Record Set and multiplies it by the current value contained in the Qty field of the selected record in the Record Set.

The second command is this:

```
RecSet.NextRecord()
```

This command uses the OpusScript Database function NextRecord to go to the next record in the Record Set called RecSet. The loop is then run again until there are no more records in the Record Set.

Summary:

This simple example, shows some three of the main things you need to include in a script when using the Database functions:

(i) First you must create a new Database object.

(ii) Then use an ExecuteSQL function to create a new

Record Set.

(iii) Use the other OpusScript Database functions with the

new Record Set.

Note:

Databases are described in the main Opus Help file. Select the Contents & Index option from the Help menu at the top of the Opus Editor. Search for the word database in the Index tab of the Help file to find topics related to the Database feature in Opus.

ExecuteSQL

Syntax:

ExecuteSQL(Command)

Return:

The Return Value varies depending upon the SQL statement executed.

For a SELECT statement the Return Value is a new Record Set object; if there is an error the Return Value is false.

For an UPDATE or INSERT statement, the Return Value is true if the action worked properly and false if it did not.

Parameters:

Command – must be a valid SQL statement. This parameter is required.

Remarks:

This function will execute a valid SQL statement, such as a SELECT or INSERT INTO command for the specified database.

Example:

To save a Record Set into a variable named Rset, use the following syntax:

```
var myDB = new Database("FILEDSN=catalogue.dsn;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products")
```

Connect

Syntax:

Connect(UserID, Password)

Return:

true if the connection is made. Otherwise, false.

Parameters:

UserID – a valid ID name for an SQL user. UserID can be a string or a variable that evaluates to a string. This parameter is optional. The system can use the default value set up in the Database tab of the Publication Properties dialog.

Password – a valid password for the UserID. Password can be a string or a variable that evaluates to a string. This parameter is optional; the system can use the default value set up in the Database tab of the Publication Properties dialog.

Remarks:

This function will connect to an SQL server. You cannot connect to an SQL server unless you have a valid user ID and password.

Example:

```
Example 1: var myDB = new Database("DSN=catalogue.dsn;") var dbConn =
```

```
myDB.Connect("JSmith", password)
```

Example 2:

For a user whose details are set in the Database tab of the Publication Properties, and with extra error checking:

```
if (myDB.Connect()) {
```

other statements here }

Commit

Syntax:

Commit()

Return:

true if the object is committed. Otherwise, false.

Remarks:

This function will commit the object to the database i.e. a record will be permanently entered on the database. For this function to work the OpusScript function AutoCommit must be false i.e. turned OFF.

```
Example: var myDB = new Database("DSN=catalogue;") myDB.AutoCommit(false) if (myDB.ExecuteSQL("INSERT INTO Product (ProdName, Price) VALUES ("Product1", 100);")) {
      if (myDB.ExecuteSQL("INSERT INTO Product (ProdName, Price) VALUES ("Product2", 200);"))
{
      myDB.Commit() // Keep the changes } else {
            myDB.Rollback() // undo the first insert if the second insert goes wrong
      } }
```

RollBack

Syntax:

Rollback()

Remarks:

This function will go back to the point in database before changes were made or the last Commit was sent. For this function to work the OpusScript function AutoCommit must be false i.e. turned OFF.

Example:

```
var myDB = new Database("DSN=catalogue;") myDB.AutoCommit(false) if
```

```
(myDB.ExecuteSQL("INSERT INTO Product (ProdName, Price) VALUES ("Product1", 100);")) {
      if (myDB.ExecuteSQL("INSERT INTO Product (ProdName, Price) VALUES ("Product2", 200);"))
{
      myDB.Commit() // Keep the changes } else {
            myDB.Rollback() // undo the first insert if the second insert goes wrong
      } }
```

AutoCommit

Syntax:

AutoCommit(Enable)

Parameters:

Enable – a boolean value. If Enable is true then AutoCommit is turned ON. Otherwise if Enable is false, AutoCommit is turned OFF. This parameter is optional. The default value is true.

Remarks:

This function will turn the AutoCommit function on or off i.e. if you UPDATE or INSERT data, the data is automatically entered in the database.

```
Example: var myDB = new Database("FILEDSN=catalogue.dsn;")
myDB.AutoCommit(false)
```

IsAutoCommitOn

Syntax:

IsAutoCommitOn()

Return:

true if AutoCommit is ON. Otherwise, false and AutoCommit is OFF.

Remarks:

This function checks if AutoCommit is on or off.

Example:

```
var myDB = new Database("DSN=catalogue;") var status = myDB.IsAutoCommitOn()
```

GetLastError

Syntax:

GetLastError()

Return:

The last database error message.

Remarks:

This function will return the last database error message sent – if any. If you did an ExecuteSQL statement and it returned an error, you could check the error message sent with this function.

Example:

```
var myDB = new Database("FILEDSN=catalogue.dsn;") if (!
MyDB.ExecuteSQL("INSERT INTO Product (ID, Type) VALUES (10,'String')")) {
      var errorMsg = myDB.GetLastError() }
```

GetIdentQuoteChar

Syntax:

GetIdentQuoteChar()

Return:

The quote character required.

Remarks:

This function will return the quote character required when quoting database table names and field names that have spaces in them. You only need to use this function when a table or field name in a database contains spaces e.g. if the table name was New Products, you would need to know the quote character required to properly reference an SQL statement.

```
Example: var myDB = new Database("DSN=catalogue;") var qType =
myDB.GetIdentQuoteChar() myDB.ExecuteSQL("INSERT INTO" + qType + "New
Products" + qType + "VALUES(10,'String')")
```

NextRecord

Syntax:

NextRecord()

Return:

The next record (if any) in the Record Set. If the Return Value is false, there are no more records in the Record Set.

Remarks:

This function will return the next record in the Record Set.

```
Example: var myDB = new Database("FILEDSN=catalogue.dsn;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") Rset.NextRecord()
```

PreviousRecord

Syntax:

PreviousRecord()

Return:

The previous record in the Record Set. If the Return Value is false, there is no previous record in the Record Set.

Remarks:

This function will return the previous record (if any) in the Record Set.

Example:

```
var myDB = new Database("FILEDSN=catalogue.dsn;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") Rset.PreviousRecord()
```

FirstRecord

Syntax:

FirstRecord()

Return:

The first record in the Record Set. If the Return Value is false, there are no records in the Record Set.

Remarks:

This function will return the first record in the Record Set.

```
Example: var myDB = new Database("DSN=catalogue;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") Rset.FirstRecord()
```

LastRecord

Syntax:

LastRecord()

Return:

The last record in the Record Set. If the Return Value is false, there are no records in the Record Set.

Remarks:

This function will return the last record in the Record Set.

Example:

```
var myDB = new Database("FILEDSN=catalogue.dsn;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") Rset.LastRecord()
```

GetRecordAt

Syntax:

GetRecordAt(Position)

Return:

The record at the specified position within the Record Set. If there is no record at that position, the Return Value is false.

Parameters:

Position – the record number within the Record Set. Position must be an integer. This parameter is required.

Remarks:

This function will return a record at the requested position within the Record Set.

```
Example: var myDB = new Database("DSN=catalogue;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") Rset.GetRecordAt(3)
```

GetRecordAtRelative

Syntax:

GetRecordAtRelative(relativePosition)

Return:

The record at the specified position relative to the current records position within the Record Set. If there is no record at that position, the Return Value is false.

Parameters:

relativePosition – the record number within the Record Set you want to go to, relative to the current record. relativePosition must be a positive or negative integer. This parameter is required.

Remarks:

This function will return a record from the Record Set relative to the current record position i.e. if the current record position was 5 and relativePosition was entered as 3, the eighth record in the Record Set would be returned.

Example:

```
var myDB = new Database("FILEDSN=catalogue.dsn;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") Rset.GetRecordAtRelative(-3) // 3
records previous to current record
```

GetNumberOfRecords

Syntax:

GetNumberOfRecords()

Return:

The number of records in the Record Set NOT in the database.

Remarks:

This function will return the number of records in the Record Set. If the record set is a sub-set of the database, the Return Value is for the Record Set and not the database.

Example:

```
var myDB = new Database("DSN=catalogue;") var Rset = myDB.ExecuteSQL("SELECT
* FROM Products") var numTotal = Rset.GetNumberOfRecords()
```

GetCurrentRecordNumber

Syntax:

GetCurrentRecordNumber()

Return:

The current record position number in the Record Set NOT in the database.

Remarks:

This function will return the current record position number in the Record Set. If the Record Set is a sub-set of the database, the Return Value is for the record set and not the database.

```
Example: var myDB = new Database("FILEDSN=catalogue.dsn;") var Rset =
myDB.ExecuteSQL("SELECT * FROM Products") var recNum =
Rset.GetCurrentRecordNumber()
```

Date - Overview

The Date object is used to represent a given date and time in a script. You must create a new Date object in the script before you can use the Date functions. There are four methods of assigning a new instance of a date

Method 1:

In this instance myDate will be set to the current date and time on your PC, for example, " Wed 21 November 2001 13:20:10". This is the most commonly used method.

```
var myDate = new Date()
```

Method 2:

In this instance myDate will be set to "31 January 2000 00:20:00", which is the number of milliseconds that have passed since Midnight 1/1/1970.

```
var myDate = new Date(949278000000)
```

Method 3:

In this instance, the date and time is entered as a string. You may enter just the date if the time is not required.

```
var myDate = new Date("Wed 21 November 2001, 13:20:10")
```

Method 4:

In this instance, the date and time is entered as individual values in the following order: Year, Month, Day, Hour, Minute. Each of the values can either be a number or a variable containing a number. In this example, the date is 31 Dec 2001 08:00:00 (

Note:

The month January is entered as the number 0 and the month December is 11.

```
var mydate = new Date(2001,11,31,8,0)
```

Get and Set Date functions:

Once you have created a new Date object in your script, you can use Get actions to retrieve the current date set in the Date object

```
var date = myDate.getDate()
```

Alternatively, once you have created a new Date object in your script, you can use the Set actions to place a new date in the Date object

```
var setDate = myDate.setDate(10)
```

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Date objects:

Class Hierarchy:

Date

Functions:

Date.parse Return the number of milliseconds since
1/1/1970

getDate Return the day of the month

getDay Return the day of the week

getFullYear Return the year

getHours Return the hour

getMilliseconds Return the number of milliseconds

getMinutes Return the minutes

getMonth Return the month

getSeconds Return the seconds

getTime Return the number of milliseconds since 1/1/1970

getTimezoneOffset Return the difference between local time
zone and universal time in minutes

getUTCDate Return the day of the month in universal time

getUTCDay Return the day of the week in universal time

getUTCFullYear Return the year in universal time

getUTCHours Return the hour in universal time

getUTCMilliseconds Return the number of milliseconds in universal time

getUTCMinutes Return the minutes in universal time

getUTCMonth Return the month in universal time

getUTCSeconds Return the seconds in universal time

setDate Set the day of the month

setFullYear Sets the year

setHours Sets the hour

setMilliseconds Sets the number of milliseconds

setMinutes Sets the minutes

setMonth Sets the month

setSeconds Sets the seconds

setTime Sets the time since 1/1/1970 in milliseconds

setUTCDate Sets the date in the month in universal time

setUTCFullYear Sets the full year in universal time

setUTCHours Sets the hour in universal time

setUTCMilliseconds Sets the number of milliseconds in universal time

setUTCMinutes Sets the number of minutes in universal time

setUTCMonth Sets the month in universal time

setUTCSeconds Sets the seconds in universal time

toGMTString Converts the Date object to a string adjusted
to GMT (Greenwich Mean Time)

toLocaleString Converts the Date object to a string adjusted to local time zone

toString Converts the Date object into a string

toUTCString Converts the Date object into a string
adjusted to universal time

valueOf Return the value of the Date object as a
number of milliseconds

getDate

Syntax:

getDate()

Return:

The day of the month. The Return Value ranges from 1 to 31.

Remarks:

This function will return the day of the month for the specified Date object.

Example:

```
var myDate = new Date() var currDate = myDate.getDate()
```

getFullYear

Syntax:

getFullYear()

Return:

The current year as a four digit number.

Remarks:

This function will return the current year as a four digit number for the specified Date object.

```
Example: var myDate = new Date() var currYear = myDate.getFullYear()
```

getMonth

Syntax:

getMonth()

Return:

The day of the month. A Return Value of 0 = Jan; 1 = Feb; 2 = Mar; 3 = Apr; 4 = May; 5 = Jun; 6 = Jul; 7 = Aug; 8 = Sept; 9 = Oct; 10 = Nov; and 11 = Dec.

Remarks:

This function will return the month for the specified Date object.

Example:

```
var myDate = new Date() var currMonth = myDate.getMonth()
```

getDay

Syntax:

getDay()

Return:

The day of the week. A Return Value of 0 = Sunday; 1 = Monday; 2 = Tuesday; 3 = Wednesday; 4 = Thursday; 5 = Friday; and 6 = Saturday.

Remarks:

This function will return the day of the week for the specified Date object.

Example:

var myDate = new Date() var currDay = myDate.getDay()

getHours

Syntax:

getHours()

Return:

The current hour set in the Date object. A Return Value of 0 = Midnight; 1 = 1am, and so on, up to 23 = 11.pm.

Remarks:

This function will return the current Hour set in the specified Date object.

Example:

```
var myDate = new Date() var currHour = myDate.getHours()
```

getMinutes

Syntax:

getMinutes()

Return:

The current number of minutes set in the Date object. The Return Value ranges from 0 to 59 minutes.

Remarks:

This function will return the current number of minutes set in the specified Date object.

Example 1:

```
var myDate = new Date() var currMins = myDate.getMinutes()
```

Example 2:

For ranges 0 to 9, only 1 digit is returned. If you want to place a leading zero in front of the digit i.e. 01 to 09 minutes, do the following:

```
if (currMins < 10) {
      currMins = "0" + currMins }
```

getSeconds

Syntax:

getSeconds()

Return:

The current number of seconds set in the Date object. The Return Value ranges from 0 to 59 seconds.

Remarks:

This function will return the current number of seconds set in the specified Date object.

```
Example 1: var myDate = new Date() var currSecs = myDate.getSeconds()
```

Example 2:

For ranges 0 to 9, only 1 digit is returned. If you want to place a leading zero in front of the digit i.e. 01 to 09 seconds, do the following:

```
if (currSecs < 10) {
  currSecs = "0" + currSecs }
```

getMilliseconds

Syntax:

getMilliseconds()

Return:

The current number of milliseconds set in the Date object.

Remarks:

This function will return the current number of milliseconds set in the specified Date object.

Example:

```
var myDate = new Date() var currMill = myDate.getMilliseconds()
```

getTime

Syntax:

getTime()

Return:

The current number of milliseconds between midnight (GMT) 1/1/1970 and the date set in the Date object.

Remarks:

This function will return a date in milliseconds. This function may be useful to compare two Date objects or the time elapsed between two dates.

```
Example: var myDate = new Date() var TimeInMill = myDate.getTime()
```

getTimezoneOffset

Syntax:

getTimezoneOffset()

Return:

The difference in minutes between the local time zone and GMT (Greenwich Mean Time) or UTC (Universal Time Code).

Remarks:

This function will return the number of minutes difference between the local time zone and GMT. Please note, the Return Value is minutes and not hours.

```
Example: var myDate = new Date() var timeZone = myDate.getTimezoneOffset()
```

Note:

Running this example in the United Kingdom will return the value 0 (zero) if the computer's time zone has been set to GMT.

getUTCDate

Syntax:

getUTCDate()

Return:

The day of the month adjusted to UTC (universal time). The Return Value ranges from 1 to 31.

Remarks:

This function will return the day of the month for the specified Date object adjusted to UTC (universal time).

Example:

```
var myDate = new Date() var currDate = myDate.getUTCDate()
```

getUTCFullYear

Syntax:

getUTCFullYear()

Return:

The current year as a four digit number adjusted to UTC (universal time).

Remarks:

This function will return the current year as a four digit number for the specified Date object adjusted to UTC (universal time).

Example:

```
var myDate = new Date() var currYear = myDate.getUTCFullYear()
```

getUTCMonth

Syntax:

getUTCMonth()

Return:

The day of the month adjusted to UTC (universal time). Possible values are: 0 = Jan; 1 = Feb; 2 = Mar; 3 = Apr; 4 = May; 5 = Jun; 6 = Jul; 7 = Aug; 8 = Sept; 9 = Oct; 10 = Nov; and 11 = Dec.

Remarks:

This function will return the month for the specified Date object adjusted to UTC (universal time).

Example:

For a Date object named myDate, use the following syntax:

```
var myDate = new Date() var currMonth = myDate.getUTCMonth()
```

getUTCDay

Syntax:

getUTCDay()

Return:

The day of the week adjusted to UTC (universal time). A Return Value of 0 = Sunday; 1 = Monday; 2 = Tuesday; 3 = Wednesday; 4 = Thursday; 5 = Friday; and 6 = Saturday.

Remarks:

This function will return the day of the week for the specified Date object adjusted to UTC (universal time).

```
Example: var myDate = new Date() var currDay = myDate.getUTCDay()
```

getUTCHours

Syntax:

getUTCHours()

Return:

The current hour set in the Date object adjusted to UTC (universal time). A Return Value of 0 = Midnight; 1 = 1am, and so on, up to 23 = 11.pm.

Remarks:

This function will return the current Hour set in the specified Date object adjusted to UTC (universal time).

Example:

```
var myDate = new Date() var currHour = myDate.getUTCHours()
```

getUTCMinutes

Syntax:

getUTCMinutes()

Return:

The current number of minutes set in the Date object adjusted to UTC (universal time). The Return Value ranges from 0 to 59 minutes.

Remarks:

This function will return the current number of minutes set in the specified Date object adjusted to UTC (universal time).

Example:

```
Example 1: var myDate = new Date() var currMins = myDate.getUTCMinutes()
```

Example 2:

For ranges 0 to 9, only 1 digit is returned. If you want to place a leading zero in front of the digit i.e. 01 to 09 minutes, do the following:

```
if (currMins < 10) {
```

```
      currMins = "0" + currMins) }
```

getUTCSeconds

Syntax:

getUTCSeconds()

Return:

The current number of seconds set in the Date object adjusted to UTC (universal time). The range is from 0 to 59 seconds.

Remarks:

This function will return the current number of seconds set in the specified Date object adjusted to UTC (universal time).

Example 1:

For a Date object named myDate, use the following syntax:

```
var myDate = new Date() var currSecs = myDate.getUTCSeconds()
```

Example 2:

For ranges 0 to 9, only 1 digit is returned. If you want to place a leading zero in front of the digit i.e. 01 to 09 seconds, do the following:

```
if (currSecs < 10) {
      currSecs = "0" + currSecs) }
```

getUTCMilliseconds

Syntax:

getUTCMilliseconds()

Return:

The current number of milliseconds set in the Date object adjusted to UTC (universal time).

Remarks:

This function will return the current number of milliseconds set in the specified Date object adjusted to UTC (universal time).

Example:

```
var myDate = new Date() var currMill = myDate.getUTCMilliseconds()
```

setDate

Syntax:

setDate(Date)

Parameters:

Date – an an integer or a variable containing an integer. x can be positive or negative. This function will set the day of the month from the start of the month. If the value is negative, it will subtract from the start of the month into the previous month and if it is greater than the number of days in the month, it will overflow into the next month. This parameter is required.

Remarks:

This function will set the day of the month for the specified Date object.

```
Example: var myDate = new Date() // assume myDate = Wed Aug 1 2001 var
currDate = myDate.setDate(3) var showDate = myDate // showDate = Fri Aug 3
2001 var currDate = myDate.setDate(-32) showDate = myDate // showDate = Fri
Jun 29 2001
```

Note:

The day of the week is automatically worked out by the system.

setFullYear

Syntax:

setFullYear(Year)

Parameters:

Year – must be a four-digit integer or a variable containing a four digit integer. Year can be any year after 1970. This parameter is required.

Remarks:

This function will set the year for the specified Date object.

```
Example: var myDate = new Date() // assume myDate = Wed Aug 1 2001 var
currDate = myDate.setFullYear(2010) var showDate = myDate // showDate = Sun
Aug 1 2010 Note:
```

The day of the week is automatically worked out by the system.

setMonth

Syntax:

setMonth(Month)

Parameters:

Month – an integer or a variable containing an integer. x can be positive or negative. The set month can be greater or less than 12. For example, if the current month is January and x is set to 12, the month will be set to January of the next year, while setting x to 13 would set the month to February of the next year. This parameter is required.

Remarks:

This function will set the month for the specified Date object.

Example:

```
var myDate = new Date() // assume myDate = Wed Aug 1 2001 var currDate =
myDate.setMonth(4) var showDate = myDate // showDate = Mon Apr 9 2001 var
currDate = myDate.setDate(-13) showDate = myDate // showDate = Sun Jan 9 2000
Note:
```

The day of the week is automatically worked out by the system.

setHours

Syntax:

setHours(Hours)

Parameters:

Hours – must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set the Hours for the specified Date object.

Example:

```
var myTime = new Date() // assume time for myTime = 10:00:00 var currTime =
myTime.setHours(3) var showTime = myDate // showTime = 03:00:00
```

setMinutes

Syntax:

setMinutes(Minutes)

Parameters:

Minutes – must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set the minutes for the specified Date object.

Example:

```
var myTime = new Date() // assume time for myTime = 10:30:00
var currTime = myTime.setMinutes(3) var showTime = myDate // showTime =
10:03:00
```

setSeconds

Syntax:

setSeconds(Seconds)

Parameters:

Seconds – must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set the seconds for the specified Date object.

```
Example: var myTime = new Date() // assume time for myTime = 10:30:00 var
currTime = myTime.setSeconds(45) var showTime = myDate // showTime = 10:30:45
```

setMilliseconds

Syntax:

setMilliseconds(Milliseconds)

Parameters:

Milliseconds – must be an integer or a variable containing an integer and should be in the range between 0 and 999. This parameter is required.

Remarks:

This function will set the milliseconds for the specified Date object.

Example:

```
var myTime = new Date() var currTime = myTime.setMilliseconds(345)
```

setUTCDate

Syntax:

setUTCDate(x)

Parameters:

x – an integer or a variable containing an integer. x can be positive or negative e.g. 32 will add 32 days to the current date; - 32 will take 32 days off the current date. This parameter is required.

Remarks:

This function will set the day of the month for the specified Date object adjusted to UTC (universal time).

```
Example: var myDate = new Date() // assume myDate = Wed Aug 1 2001 var
currDate = myDate.setUTCDate(3) var showDate = myDate // showDate = Fri Aug 3
2001 var currDate = myDate.setUTCDate(-32) showDate = myDate // showDate =
Fri Jun 29 2001 Note:
```

The day of the week is automatically worked out by the system.

setUTCFullYear

Syntax:

setUTCFullYear(Year)

Parameters:

Year – must be a four-digit integer or a variable containing a four- digit integer. Year can be any year after 1970. This parameter is required.

Remarks:

This function will set the year for the specified Date object adjusted to UTC (universal time).

```
Example: var myDate = new Date() // assume myDate = Wed Aug 1 2001 var
currDate = myDate.setUTCFullYear(2010) var showDate = myDate // showDate =
Sun Aug 1 2010 Note:
```

The day of the week is automatically worked out by the system.

setUTCMonth

Syntax:

setUTCMonth(Month)

Parameters:

Month – an integer or a variable containing an integer. x can be positive or negative e.g. 13 will add 13 months to the current date; -13 will take 13 months off the current date. This parameter is required.

Remarks:

This function will set the month for the specified Date object adjusted to UTC (universal time).

Example:

```
var myDate = new Date() // assume myDate = Wed Aug 1 2001 var currDate =
myDate.setUTCMonth(4) var showDate = myDate // showDate = Mon Apr 9 2001 var
currDate = myDate.setUTCDate(-13) showDate = myDate // showDate = Sun Jan 9
2000 Note:
```

The day of the week is automatically worked out by the system.

setUTCHours

Syntax:

setUTCHours(Hours)

Parameters:

Hours – must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set the Hours for the specified Date object adjusted to UTC (universal time).

```
Example: var myTime = new Date() // assume time for myTime = 10:00:00 var
currTime = myTime.setUTCHours(3) var showTime = myDate // showTime = 03:00:00
```

setUTCMinutes

Syntax:

setUTCMinutes(Minutes)

Parameters:

Minutes – must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set the minutes for the specified Date object adjusted to UTC (universal time).

Example:

```
var myTime = new Date() // assume time for myTime = 10:30:00 var currTime =
myTime.setUTCMinutes(3) var showTime = myDate // showTime = 10:03:00
```

setUTCSeconds

Syntax:

setUTCSeconds(Seconds)

Parameters:

Seconds – must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set the seconds for the specified Date object adjusted to UTC (universal time).

```
Example: var myTime = new Date() // assume time for myTime = 10:30:00 var
currTime = myTime.setUTCSeconds(45) var showTime = myDate // showTime =
10:03:45
```

setUTCMilliseconds

Syntax:

setUTCMilliseconds(Milliseconds)

Parameters:

Milliseconds – must be an integer or a variable containing an integer and should be in the range between 0 and 999. This parameter is required.

Remarks:

This function will set the milliseconds for the specified Date object adjusted to UTC (universal

time).

Example:

```
var myTime = new Date() var currTime = myTime.setUTCMilliseconds(345)
```

setTime

Syntax:

setTime(Milliseconds)

Parameters:

Milliseconds – the number of milliseconds between the required date and time and midnight (GMT) 1/1/1970. Millseconds must be an integer or a variable containing an integer. This parameter is required.

Remarks:

This function will set a date in milliseconds for the specified Date object. This function is often used with the Date.parse function to assign a date value from a string.

```
Example: var newTime = new Date() newTime.setTime(Date.parse("Aug 1, 2001"))
Note:
```

In this example, the Date.parse function converts a string into milliseconds, which the setTime function then uses to set the time for the Date object newTime. The Date object newTime is now set to Aug 1 2001.

toGMTString

Syntax:

toGMTString()

Return:

A specified date converted from the local time zone to the GMT time zone before being converted to a string.

Remarks:

This function will convert the Date object to a string adjusted to GMT (Greenwich Mean Time).

Example:

```
var myDate = new Date() var dateString = myDate.toGMTString()
```

see also: Date.parse, toLocaleString, toUTCString

toLocaleString

Syntax:

toLocaleString()

Return:

A date converted to a string using the local time zone.

Remarks:

This function will convert a Date object to a string using the local time zone.

Example:

```
var myDate = new Date() var dateString = myDate.toLocaleString()
```

toString

Syntax:

toString()

Return:

A date converted to a string using the local time zone.

Remarks:

This function will convert a Date object to a string using the local time zone.

```
Example: var myDate = new Date() var dateString = myDate.toString()
```

toUTCString

Syntax:

toUTCString()

Return:

A date converted to a string adjusted to UTC (universal time).

Remarks:

This function will convert a Date object to a string adjusted to UTC (universal time).

Example:

```
var myDate = new Date() var dateString = myDate.toUTCString()
```

valueOf

Syntax:

valueOf()

Return:

A date converted to a number and the number is the same number of milliseconds as returned

by the getTime function.

Remarks:

This function will convert a Date object to a number.

```
Example: var myDate = new Date() var dateNum = myDate.valueOf()
```

Date.parse

Syntax:

Date.parse(Date)

Parameters:

Date – a string containing the date and time to be parsed. This parameter is required.

Remarks:

This method will parse the date contained in the string and return a value in milliseconds, which can then be used to create a new Date object or used to set the date in an existing Date object with the setTime function.

```
Example: var myTime = new Date() var parseDate = Date.parse("Mon, 19 Nov 2001
17:41:46 –0400")
```

Note:

In this example, the date is set by entering a string. The –0400 part of the string refers to the timezone.

Math - Overview

The Math object allows you to perform a variety of scientific calculations in your script including functions for logarithms, trigonometry and geometry.

The functions listed below are all part of the Math object and are either (i) Math constants, such as PI; (ii) or Math methods, such as acos. This means that in the script they must be typed in as follows:

Math.functionName

This means, type the word Math followed by the name of one of the OpusScript functions listed below separated by a full stop, For example:

```
var myMath = Math.PI var myMath = Math.acos(-0.01234)
```

Note:

Some of the Math functions have a constant value, such as, Math.PI because the value of Pi never changes. Other Math functions will evaluate an answer based on parameters you enter, such as, the Math.acos function – see Math for more information.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Math objects:

Class Hierarchy:

Math

Functions:

Math.abs Returns the absolute value of a number

Math.acos Returns the arccosine of a number (in radians)

Math.asin Returns the arcsine of a number (in radians)

Math.atan Returns the arctangent of a number (in radians)

Math.atan2 Returns the angle from the X-axis to a point

Math.ceil Rounds a number up

Math.cos Returns the cosine of a number

Math.E Returns Euler's constant e (approx. 2.718)

Math.exp Returns E to the power of the argument passed

Math.floor Rounds a number down

Math.LN10 Returns the natural log of 10 (approx. 2.303)

Math.log Returns the natural log (base E) of a number

Math.LOG10E Returns the base 10 log of E (approx. 0.434)

Math.max Returns the larger of two values

Math.min Returns the smaller of two values
Math.PI Returns PI (approx. 3.1415)
Math.pow Returns the first parameter raised to the power of the second
Math.random Returns a pseudo-random number between 0 and 1
Math.round Rounds to the nearest integer
Math.sin Returns the sine of a number
Math.sqrt Returns the square root of a number
Math.SQRT1_2 Returns the square root of ½ (approx. 0.707)
Math.SQRT2 Returns the square root of 2 (approx. 1.414)
Math.tan Returns the tangent of a number

Math.E

Syntax:

Math.E

Constant:

Euler’s constant i.e. the base of natural logarithms (approx. 2.718).

Example:

```
var myMath = Math.E
```

Math.LN10

Syntax:

Math.LN10

Constant:

The natural logarithm of 10 (approx. 2.302).

```
Example: var myMath = Math.LN10
```

Math.LOG10E

Syntax:

Math.LOG10E

Constant:

The base 10 logarithm of E (approx. 0.434).

```
Example: var myMath = Math.LOG10E
```

Math.LOG2E

Syntax:

Math.LOG2E

Constant:

The base 2 logarithm of E (approx. 1.442).

Example:

```
var myMath = Math.LOG2E
```

Math.PI

Syntax:

Math.PI

Constant:

PI i.e. the ratio of the circumference of a circle to its diameter (approx. 3.1415).

```
Example: var myMath = Math.PI
```

Math.SQRT1_2

Syntax:

Math.SQRT1_2

Constant:

The sqaure root of 1⁄2 (approx. 0.707).

```
Example2 var myMath = Math.SQRT1_2
```

Math.SQRT2

Syntax:

Math.SQRT2

Constant:

The sqaure root of 2 (approx. 1.414).

```
Example var myMath = Math.SQRT2
```

Math.abs

Syntax:

Math.abs(x)

Return:

The absolute (i.e. positive) value of a number.

Parameters:

x – the number or a variable containing a number. This parameter is required.

```
Example var myMath = Math.abs(-0.01234)
```

Math.acos

Syntax:

Math.acos(x)

Return:

The arc-cosine of a number in radians.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example:

```
var myMath = Math.acos(-0.01234)
```

Math.asin

Syntax:

Math.asin(x)

Return:

The arc-sine of a number in radians.

Parameters:

x – the number or a variable containing a number. This parameter is required.

```
Example: var myMath = Math.asin(0.234)
```

Math.atan

Syntax:

Math.atan(x)

Return:

The arc-tangent of a number in radians.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example:

```
var myMath = Math.atan(0.234)
```

Math.atan2

Syntax:

Math.atan2(x, y)

Return:

The angle in radians of the point from the y-axis.

Parameters:

x – a number or a variable containing a number. This parameter is required.

y – a number or a variable containing a number. This parameter is required.

```
Example: var myMath = Math.atan2(23,45)
```

Math.ceil

Syntax:

Math.ceil(x)

Return:

The value of a number rounded up to the nearest integer.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example:

Example 1:

```
var myMath = Math.ceil(23.45678) // returns 24
```

Example 2:

```
var myMath = Math.ceil(-9.999) // returns -9
```

Math.cos

Syntax:

Math.cos(x)

Return:

The cosine of a number.

Parameters:

x – the number or a variable containing a number. The number is the angle in radians and not degrees. This parameter is required.

```
Example: var myMath = Math.cos(23)
```

Math.exp(x)

Syntax:

Math.exp(x)

Return:

E to the power of x.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example:

```
var myMath = Math.exp(3)
```

Math.floor

Syntax:

Math.floor(x)

Return:

The value of a number rounded down to the nearest integer.

Parameters:

x – the number or a variable containing a number. This parameter is required.

```
Example 1: var myMath = Math.floor(23.789) // returns 23
Example 2: var myMath = Math.floor(-9.999) // returns -10
```

Math.log

Syntax:

Math.log(x)

Return:

The natural logaritm of a number.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example:

```
var myMath = Math.log(23.789)
```

Math.max

Syntax:

Math.max(x, y)

Return:

The larger number of the two values entered in x and y.

Parameters:

x – the number or a variable containing a number. This parameter is required.

y – the number or a variable containing a number. This parameter is required.

```
Example: var Score1 = 75 var Score2 = 65 var myMath = Math.max(Score1,
Score2) // returns 75
```

Math.min

Syntax:

Math.min(x, y)

Return:

The smaller number of the two values entered in x and y.

Parameters:

x – the number or a variable containing a number. This parameter is required.

y – the number or a variable containing a number. This parameter is required.

```
Example: var Score1 = 75 var Score2 = 65 var myMath = Math.min(Score1,
Score2) // returns 75
```

Math.pow

Syntax:

Math.pow(x, y)

Return:

The x raised to the power of y.

Parameters:

x – the number or a variable containing a number. This parameter is required.

y – the number or a variable containing a number. This parameter is required.

```
Example var myMath = Math.pow(2, 8) // returns 256
```

Math.random

Syntax:

Math.random()

Return:

A pseudo-random number between 0 and 1.

Remarks:

Computer generated numbers are not truly random. However for most general purposes the pseudo-random numbers are more than sufficient. This only affects mathematical theory, not

real- world use. No parameters are required.

Examples:

```
Example 1: var myMath = Math.random()
```

Example 2: To get a random number between 1 and 10:

```
var myRandom = Math.round(Math.random()*10)
```

Math.round

Syntax:

Math.round(x)

Return:

x rounded up or down to the nearest integer.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example 1:

```
var myMath = Math.round(2.345) // returns 2
```

Example 2:

```
var myMath = Math.round(2.567) // returns 3
```

Math.sin

Syntax:

Math.sin(x)

Return:

The sine of x.

Parameters:

x – the number or a variable containing a number. The number is the angle in radians and not degrees. This parameter is required.

```
Example: var myMath = Math.sin(2.345)
```

Math.sqrt

Syntax:

Math.sqrt(x)

Return:

The square root of x.

Parameters:

x – the number or a variable containing a number. This parameter is required.

Example:

```
var myMath = Math.sqrt(2.345)
```

Math.tan

Syntax:

Math.tan(x)

Return:

The tangent of x.

Parameters:

x – the number or a variable containing a number. The number is the angle in radians and not degrees. This parameter is required.

```
Example: var myMath = Math.tan(45)
```

Event - Overview

Any object in a publication that can have actions added to it can use the Event functions listed below. These functions allow you to create a trigger from within a script rather than having to use the Triggers in an object's Actions dialog.

Note:

Browser objects are the only type of object that cannot use these Event functions.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Event objects:

Class Hierarchy:

Event

Functions:

RegisterEventHandler Set a function to be called when a certain

event triggers on the object.

TriggerComponentEvent Activate a Custom Trigger for a specified

object.

UnregisterEventHandler Remove a previously registered event

handler.

RegisterEventHandler

Syntax:

RegisterEventHandler(EventName, EventFunction)

Return:

An identifier for the added event handler, which may be used to remove the Event Handler at a later time using the UnRegisterEventHandler function

Parameters:

EventName – the name of the event that the function is to handle. EventName must be surrounded by quote marks and must be one of the events described below – see Remarks for more information. This parameter is required.

EventFunction – a function that will handle the event. The function should take one argument that will be an object containing information about the event that triggered the call. This parameter is required.

Remarks:

There are a variety of EventNames that can be passed to the EventFunction, these fall into two categories:

(i) Mouse Events – events that happen when the user

presses a button on a mouse.

(ii) Keyboard Events – events that happen when the user

types a key on the keyboard.

Mouse Events:

Event Name Description

mouseover mouse moves over the object.

mouseout mouse moves off the object.

lmousedown left mouse button is pressed down.

lmouseup left mouse button is released.

lclick left mouse button is clicked once over the

object.

ldblclick left mouse is double-clicked over the object.

rmousedown right mouse button is pressed down.

rmouseup right mouse button is released.

rclick right mouse button is clicked once over the object.

rdblclick right mouse button is double-clicked over the object.

mousemove the mouse is moved.

mousewheelup the mouse moved up.

mousewheeldown the mouse moved down.

The propertiesof the event object of Mouse Events are:

x x coordinate of the mouse, relative to the top

left corner of the page.

y y coordinate of the mouse, relative to the top

left corner of the page.

Keyboard Events:

Event Name Description

keydown a key was pressed down.

keyup a key was released.

keypress a key was pressed generating a character

The properties of the event object of Keyboard Events are:

key the character of the key that was pressed.

autorepeat the number of times the event has autorepeated

i.e. 0 (zero) for the first time.

Example 1:

In this example, a page contains two radio buttons named MoveButSlow and moveButFast and one image named image1. This script was written in a Script Action. Whenever, image1 is left mouse clicked, the function Move is called and depending on which radio button is currently pressed down, the image will move to its new position immediately or over a two second period

```
var myEvent = image1.RegisterEventHandler("lclick", this.Move)
function Move() {
if(moveButSlow.GetState() == true) {
      image1.Move(180,200,2) } if (moveButFast.GetState() == true) {
      image1.Move(180,200) } }
```

Example 2:

Below is an example of the RegisterEventHandler using a propertyof the Mouse Event

```
var myEvent = image1.RegisterEventHandler("lclick", this.Move)
function Move(property) {
var newXpos = property.x var newYPos = property.y
Text.SetPosition(property.x,property.y,2.784) }
```

UnregisterEventHandler

Syntax:

UnregisterEventHandler(EventID)

Parameters:

EventID – the Identifier returned by the RegisterEventHandler function when the handler was registered. This parameter is required.

Example:

Below is an example of UnRegisterEventHandler:

```
var myEvent = image1.RegisterEventHandler("lclick", this.Move) // other
statements here image1.UnRegisterEventHandler(myEvent)
```

TriggerComponentEvent

Syntax:

TriggerComponentEvent(CustomTrigger)

Parameters:

CustomTrigger – the name of the Custom Trigger to be activated. This parameter is required.

Remarks:

This function will activate a Custom Trigger for a specified object. The Custom Trigger must have already been specified prior to calling it with the TriggerComponentEvent function.

Example:

For a Button object named myButton, use the following syntax to activate the Custom Trigger startProcess

```
myButton.TriggerComponentEvent("startProcess")
```

MultiFrames - Overview

The MultiFrame functions allow you to use a script to display different frames of a MultiFrame object on a page in your publication. These functions are equivalent to the MultiFrame actions in the MultiFrame menu of the Actions dialog.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

MultiFrames

Basic Objects

Functions:

Backward Go back one frame

Forward Go forward one frame

ToEnd Go to last frame in the MultiFrame

ToFrame Go to a given frame in the MultiFrame

ToStart Go to first frame in the MultiFrame

Graphical Objects

Forward

Syntax:

Forward()

Remarks:

This function will go forward one frame in a specified MultiFrame object. No parameters required.

Example:

For a MultiFrame object named MultiFrame 1, use the following syntax:

```
MultiFrame_1.Forward()
```

Backward

Syntax:

Backward()

Remarks:

This function will go backward one frame in a specified MultiFrame object. No parameters required.

Example:

For a MultiFrame object named MultiFrame 1, use the following syntax:

```
MultiFrame_1.Backward()
```

ToStart

Syntax:

ToStart()

Remarks:

This function will go to the starting frame in a specified MultiFrame object. No parameters required.

Example:

For a MultiFrame object named MultiFrame 1, use the following syntax:

```
MultiFrame_1.ToStart()
```

ToEnd

Syntax:

ToEnd()

Remarks:

This function will go to the ending frame in a specified MultiFrame object. No parameters required.

Example:

For a MultiFrame object named MultiFrame 1, use the following syntax:

```
MultiFrame_1.ToEnd()
```

ToFrame

Syntax:

ToFrame(Number)

Parameters:

Number – The index position number of the frame in a specified MultiFrame. Number is a zero-based index. This parameter is required.

Remarks:

This function will go to a specific frame in a specified MultiFrame object.

Example:

For a MultiFrame object named MultiFrame 1, to go to the second frame, use the following syntax:

```
MultiFrame_1.ToFrame(1)
```

Search - Overview

The Search functions allow you to create custom search dialogs.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

Search

Basic Objects

Functions:

GetFirstKeyword Gets first keyword in search list

GetFirstPage Gets first page to search on

GetNextKeyword Gets next keyword in search list

GetNextPage Gets next page to search on

OpenSearch Returns a search object containing the publication search words

GetFirstKeyword

Syntax:

GetFirstKeyword()

Return:

The first keyword in the specified Search object. If there is no first keyword, the value returned will be null.

Remarks:

This function will return the first keyword in a specified Search object. To find the first keyword in the publication search, you must first create a Search object using OpenSearch Note:

This function also sets an internal value for the Search object back to the first entry in the list, in order to recursively get entries out of the search list.

Example:

```
var Search = OpenSearch() var Keyword = Search.GetFirstKeyword()
```

GetNextKeyword

Syntax:

GetNextKeyword()

Return:

The next keyword in the specified Search object. If there is no next keyword, the value returned will be null.

Remarks:

This function will return the next keyword in a specified Search object. To find the next keyword in the publication search, you must first create a Search object using OpenSearch

Note: This function also sets an internal value for the Search object, in order to recursively get entries out of the search list.

Example:

```
var Search = OpenSearch() var Keyword = Search.GetFirstKeyword() while (true)
{
if (Keyword == "") {
      break } Keyword = Search.GetNextKeyword() }
```

GetFirstPage

Syntax:

GetFirstPage(Keyword)

Return:

The first Page object that contains a specified keyword.

If the keyword is not found on any pages the return is NULL.

Remarks:

This function will return the first Page object that contains a specified keyword in a Search object. To find a Page object, you must first create a Search object using OpenSearch.

```
Example: var Search = OpenSearch() var Keyword = Search.GetFirstKeyword() var Page = Search.GetFirstPage(Keyword) GotoPage(Page) // go to the page in the search
```

GetNextPage

Syntax:

GetNextPage(Keyword)

Return:

The next Page object that contains a specified keyword in a Search object.

If the keyword is not found on any pages the return is NULL.

Remarks:

This function will return the next Page object that contains a specified keyword in a Search object. To find a Page object, you must first create a Search object using OpenSearch.

Example:

To create a list of all pages with the keyword "Opus", use the following syntax:

```
var Search = OpenSearch() var Keyword = Search.GetFirstKeyword() while (true) {
if (Keyword == "Opus") {
var Page = Search.GetFirstPage(Keyword) while (true) {
if (Page == NULL) {
      break }
AddToOutputPageDisplay(Page) Page = Search.GetNextPage(Keyword) } } Keyword = Search.GetNextKeyword }
```

Number - Overview

The Number object allows you to perform mathematical calculations in your script. These functions are all methods or properties of the Number object, which means that in the script they must be called as follows:

```
Number.functionName
```

This means, type the word Number followed by the name of one of the OpusScript functions listed below separated by a full stop, For example:

```
Number.MAX_VALUE Number.toString()
```

Note: Some of the Number functions have a constant value, such as, Number.MAX_VALUE because the value never changes. Other Number functions will evaluate an answer, such as, the Number.NaN function.

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Number objects:

Class Hierarchy:

Number

Functions:

MAX_VALUE Returns the largest number that can be

represented in OpusScript

MIN_VALUE Returnts the smallest number that can be

represented in OpusScript

NaN Returns a value that is "not a number"

NEGATIVE_INFINITY Returns a value representing negative

infinity

POSITIVE_INFINITY Returns a value representing positive

infinity

toString Converts the Number object into a string

valueOf Returns the primitive value of the Number

Object

Number.MAX_VALUE

Syntax:

Number.MAX_VALUE

Constant:

The largest number that can be represented by OpusScript (approximately 1.79E+308). Numbers larger than MAX_VALUE are represented as ‘infinity’.

```
Example: if (num1 * num2 <= Number.MAX_VALUE) {
      doMaths() } else {
      newNums() }
```

Number.MIN_VALUE

Syntax:

Number.MIN_VALUE

Constant:

The smallest number that can be represented by OpusScript (approximately 2.22E-308). Numbers smaller than MIN_VALUE are represented as 0.

Example:

```
if (num1 * num2 >= Number.MIN_VALUE) {
      doMaths() }
else {
      newNums() }
```

Number.NaN

Syntax:

Number.NaN

Constant:

A value that is ‘not a number’.

```
Example: var month = 13 if (month < 1 || month > 12) {
month = Number.NaN msgText.ReplaceSelection("Month must be between 1 and
12.") }
```

Number.NEGATIVE_INFINITY

Syntax:

Number.NEGATIVE_INFINITY

Constant:

A value representing negative infinity.

```
Example: var smallNumber = -Number.MAX_VALUE * 10 if (smallNumber ==
Number.NEGATIVE_INFINITY) {
      firstFunction() }
else {
      secondFunction() }
```

Number.POSITIVE_INFINITY

Syntax:

Number.POSITIVE_INFINITY

Constant:

A value representing positive infinity.

```
Example: var bigNumber = Number.MAX_VALUE * 10 if (bigNumber ==
Number.POSITIVE_INFINITY) {
      firstFunction() } else {
      secondFunction() }
```

toString

Syntax:

toString()

Return:

A string.

Remarks:

This function converts a number into a string.

Example:

For a Text Input object named Text_1, use the following syntax:

```
var answer = 75 Text_1.ReplaceSelection("Your answer was: " +
answer.toString())
```

valueOf

Syntax:

valueOf()

Return:

The primitive value of the Number object.

```
Example: var myNum = new Number(75) var numValue = myNum.valueOf()
```

String - Overview

The String object allows you to manipulate strings (i.e. alphanumeric characters surrounded by quite marks in a variable) in your string. These functions are all methods or properties of the String object, which means that in the script they must be called as follows:

```
String.functionName
```

This means, type the word String followed by the name of one of the OpusScript functions listed below separated by a full stop, For example:

```
String.charAt(3)
```

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with String objects:

Class Hierarchy:

String

Functions:

charAt Get the character at the specified position in the

string

charCodeAt Get the character code at the specified position in

the string

indexOf Return the position within the String object of

the substring

lastIndexOf Return the position within the String object of

the last match in the substring

length The length of the string

split Split a String object into an array of strings

substring Returns a substring of a specified string

toLowerCase Converts the String object to lower case

toString Converts the String object into a string

toUpperCase Converst the String object to upper case

valueOf Returns the primative value of the String object

charAt

Syntax:

charAt(index)

Return:

The character at the specifed index position.

Parameters:

index – a number or a variable containing a number. Index is zero-based index. This parameter is optional and will start at the beginning of the string if not entered.

Example:

```
var myWord = new String("Opus") myWord.charAt(2) // will return the character "u"
```

charCodeAt

Syntax:

charCodeAt(index)

Return:

The ISO-Latin-1 codeset of the specified character at the index position. The codeset ranges from 0 to 255. The codes 0 to 127 are a direct match of the ASCII character set.

Parameters:

index – a number or a variable containing a number. Index is zero-based index. This parameter is optional and will start at the beginning of the string if not entered.

```
Example: var myWord = new String("Opus") myWord.charCodeAt(2) // will return the codeset character for "u" of 117
```

indexOf

Syntax:

indexOf(substring, index)

Return:

The character position in the string at which the substring was found. If the substring is not found the return value is -1.

Parameters:

substring – the substring to search for in the string. Substring can be a string or a variable name containing a string. Substring should be surrounded by quotes if it is a string. This parameter is required.

index – the index position to start searching from. Index can be a number or a variable containing a number. Index is a zero-based index. This parameter is optional and will start at the beginning of the string if not entered.

Remarks:

This function will search for the substring starting at the beginning of the string (or the index position you specifed) and work towards the end of the string.

```
Example: var myWord = new String("John Doe, John Smith")
myWord.indexOf("John") // will return 0 (zero) as the index postion
myWord.indexOf("John",4) // will return 10 as the index position
```

lastIndexOf

Syntax:

lastIndexOf(substring, index)

Return:

The character position in the string at which the substring was found. If the substring is not found the return value is -1.

Parameters:

substring – the substring to search for in the string. Substring can be a string or a variable name containing a string. Substring should be surrounded by quotes if it is a string. This parameter is required.

index – the index position to start searching from. Index can be a number or a variable containing a number. Index is a zero-based index. This parameter is optional and will start at the end of the string if not entered.

Remarks:

This function will search for the substring starting at the end of the string (or the index position you specifed) and work towards the beginning of the string.

Example:

```
var myWord = new String("John Doe, John Smith") myWord.lastIndexOf("John") //
will return 10 as the index postion
```

split

Syntax:

split(separator, limit)

Return:

A new array.

Parameters:

separator – the character to use to separate the string. This parameter is optional and the return value will be one array element containing the entire string.

limit – an integer specifying a limit on the number of splits to be found. This parameter is optional.

Remarks:

This function will split a string into substrings by a specified separator. Each new substring is an element in the array.

Example:

```
var myWord = new String("Welcome to Digital Workshop") var myArray =
myWord.split(" ") // the separator is a space
myText.ReplaceSelection(myArray[2]) // element [2] of the array is "Digital"
```

length

Syntax:

length

Remarks:

length is a property of a string that contains the number of characters in the specified string.
Note:

When using length, you should not append length with round brackets.

```
Example: var welcome = "Hello" strLength = welcome.length
```

Note:

In this example the variable strLength will contain the number 5. Please note when using length, you should not append length with round brackets.

valueOf

Syntax:

valueOf()

Return:

The primitive value of the string Object.

```
Example: var myString = new String(10) myString.valueOf()
```

toString

Syntax:

toString()

Return:

A string.

Remarks:

This function converts a String object into a string.

Example:

For a Text Input object named Text_2, use the following syntax:

```
var answer = new String ("Room") answer += 101 Text_2.ReplaceSelection("Your answer was: " + answer.toString())
```

substring

Syntax:

substring(From, To)

Return:

A substring of a string.

Parameters:

From – an integer specifying the position within the string of the first character of the substring. The index position of the first character of a string is 0. This parameter is required.

To – An integer specifying a position within the string that is one greater than the last character of the substring. In other words, the position of the character specified in To is not included in the substring. This parameter is optional; the default value is -1 i.e. the end of the string.

Example:

```
var myString = "Hello James Smith" var mySub = myString.substring(0,5) // mySub will contain the value 'Hello' var mySub = myString.substring(6,11) // mySub will contain 'James' var mySub = myString.substring(6) // mySub will contain "James Smith'
```

toLowerCase

Syntax:

toLowerCase()

Return:

The specified a variable's string with all upper case letters converted to lower case.

```
Example: var surname = "Smith" surname = surname.toLowerCase() // the variable surname will contain the value "smith"
```

toUpperCase

Syntax:

toUpperCase()

Return:

The specified a variable's string with all upper case letters converted to lower case.

```
Example: var surname = "Smith" surname = surname.toUpperCase() // the
variable surname will contain the value "SMITH"
```

Boolean - Overview

The Boolean Object can only have a value of true or false.

Any of the following methods can be used to create a new Boolean object with an initial value of FALSE:

```
var bFalse = new Boolean() var bFalse = new Boolean(0) var bFalse = new Boolean(null) var bFalse = new Boolean("") var bFalse = new Boolean(false)
```

Any of the following methods can be used to create a new Boolean object with an initial value of TRUE:

```
var bTrue = new Boolean(true) var bTrue = new Boolean(1) var bTrue = new Boolean("true") var bTrue = new Boolean("false") var bTrue = new Boolean("any string")
```

Hierarchy:

Only the specific functions listed under the Functions: heading below can be used with Basic objects:

Class Hierarchy:

String

Functions:

toString Converts a Boolean object to a string

valueOf Returns a primitive value of a Boolean Object

toString

Syntax:

toString()

Return:

A string.

Remarks:

This function converts a Boolean object into a string.

```
Example: var answer = new Boolean(true) if (answer.toString() == "true") {
```

other statements here }

valueOf

Syntax:

valueOf()

Return:

The primitive value of the Boolean object.

Example:

```
var answer = new Boolean(true) if (answer.valueOf() == true) {
      other statements here }
```

QuickTime VR - Overview

The QuickTime VR functions allow you to use a script to control a QuickTime VR object on a page in your publication. For example, you can use the functions QuickTime VR to pan or zoom the current view.

For more information on QuickTime VR see the Apple QuickTime Authoring website or use an internet search engine.

Hierarchy:

As well as the specific functions listed under the Functions: heading below, you can also use the following OpusScript functions:

Class Hierarchy:

QuickTime VR

Basic Objects

Functions:

EnableHotspot Enable or disable one, some or all of the

hotspots in a QuickTime VR object.

GetCurrentNode Get the current object Node ID of a

QuickTime VR object.

GetFieldOfView Get the current field of view ("zoom")

value of a QuickTime VR object

GetHotspotType Get the type of a QuickTime VR hotspot.

GetPanAngle Get the current pan angle of a QuickTime

VR object (in degrees)

GetTiltAngle Get the current tilt angle of a QuickTime

VR object (in degrees)

GetViewingLimits Get the current viewing limits of a

QuickTime VR object.

GetVisibleHotspots Get an array of currently visible hotspots in

a QuickTime VR object.

SetFieldOfView Set the current field of view ("zoom") value

of a QuickTime VR object

SetHotspotCallback Set a script function to be called when a hotspot is clicked in the QuickTime VR object.

SetMouseOverCallback Set a script function to be called when the mouse cursor is over a hotspot in the QuickTime VR object.

SetPanAngle Set the current pan angle of a QuickTime

VR object (in degrees)

SetTiltAngle Set the current tilt angle of a QuickTime

VR object (in degrees)

GetPanAngle

Syntax:

GetPanAngle()

Return:

Current pan angle in degrees

Remarks:

This function will get the current pan (left-rght) angle of the QuickTimeVR object it is called on.

Example:

To get the current pan angle of a QuickTime VR object called qtvr:

```
var Pan = qtvr.GetPan()
```

SetPanAngle

Syntax:

SetPanAngle(Angle)

Parameters:

Angle – new pan angle. Maximum range of 0 to 360. This parameter is required.

Remarks:

This function will set the pan (left-rght) angle of the QuickTimeVR object it is called on.

The range of possible angles is set by the author of the QuickTimeVR movie. Any value outside the valid range will be clipped to the valid range.

Example:

To pan a QuickTime VR object called qtvr in a complete circle:

```
for (angle = 0; angle <= 360; angle += 0.2) {
qtvr.SetPanAngle( angle ) wait( 0 ) }
```

Note: This example assumes that the pan angle range specifed in the QuickTime VR movie is from 0 to 360. This may not be the case for all QuickTime VR movies and will depend on the movie author.

GetTiltAngle

Syntax:

GetTiltAngle()

Return:

Current tilt angle in degrees

Remarks:

This function will get the current tilt (up-down) angle of the QuickTimeVR object it is called on.

Example:

To get the current tile angle of a QuickTime VR object called qtvr:

```
var Tilt = qtvr.GetTilt()
```

SetTiltAngle

Syntax:

SetTiltAngle(Angle)

Parameters:

Angle – new tilt angle. Maximum range of -90 to 90, where 0 represents the horizontal. This parameter is required.

Remarks:

This function will set the tilt (up-down) angle of the QuickTimeVR object it is called on.

The range of possible angles is set by the author of the QuickTimeVR movie. Any value outside the valid range will be clipped to the valid range.

Example:

To tilt a QuickTime VR object called qtvr from vertically down through to vertically up:

```
for (angle = -90; angle <= 90; angle += 0.2) {
qtvr.SetTiltAngle( angle ) wait( 0 ) }
```

Note: This example assumes that the tilt angle range specifed in the QuickTime VR movie is from -90 to 90. This may not be the case for all QuickTime VR movies and will depend on the movie author.

GetFieldOfView

Syntax:

GetFieldOfView()

Return:

Current field of view.

Remarks:

This function will get the current field of view (zoom level) of the QuickTimeVR object it is called on.

Example:

To get the current field of view of a QuickTime VR object called qtvr:

```
var FOV = qtvr.GetFieldOfView()
```

SetFieldOfView

Syntax:

SetFieldOfView(FOV)

Parameters:

FOV – New field of view. The range is dependent on the author of the QuickTime VR file. 1.0 is the initial field of view. This parameter is required.

Remarks:

This function will set the field of view of the QuickTimeVR object it is called on.

The range of possible angles is set by the author of the QuickTimeVR movie. Any value outside the valid range will be clipped to the valid range.

Example 1:

To reset the Field of View of a QuickTime VR object called qtvr to the original default

```
qtvr.SetFieldOfView( 1.0 )
```

Example 2:

To change the Field of View of a QuickTime VR object called qtvr:

```
for (FOV = 0.9; FOV < 1.1; FOV += 0.05) {
qtvr.SetFieldOfView( FOV ) wait( 0 ) }
```

Note: This example assumes that the Field of View range specifed in the QuickTime VR movie is from 0.9 to 1.1. This may not be the case for all QuickTime VR movies and will depend on the movie author.

GetViewingLimits

Syntax:

GetViewingLimits(Type)

Return:

A new Object with the following properties:

fMin - the minimum value allowed for the given range.

fMax - the maximum value allowed for the given range.

Parameters:

Type – an integer specifying the type of limits to be retrieved. This parameter is required.

Note:

The possible values for Type are:

Value of Type

Range to retrieve

0 Pan angle

1 Tilt angle

2 Field of View

Remarks:

This function will get the range of values allowed for a given range type of the QuickTimeVR object it is called on.

Example:

This example gets the Pan, Tilt and Field of View ranges for a QuickTime VR object called qtvr

```
var Pan_Range = qtvr.GetViewingLimits(0); Debug.trace( "Pan: fMin = " +
Pan_Range.fMin + " fMax = " + Pan_Range.fMax + "\n")
```

```
var Tilt_Range = qtvr.GetViewingLimits(1); Debug.trace( "Tilt: fMin = " +
Tilt_Range.fMin + " fMax = " + Tilt_Range.fMax + "\n")
```

```
var FoV_Range = qtvr.GetViewingLimits(2); Debug.trace( "FoV: fMin = " +
FoV_Range.fMin + " fMax = " + FoV_Range.fMax + "\n")
```

SetHotspotCallback

Syntax:

SetHotspotCallback(HSCallback)

Parameters:

HSCallback – A function to call when a hotspot is triggered in the QuickTime VR object. The function must take one parameter. This parameter is required.

Remarks:

This function will set a callback script function to be called when a hotspot is clicked on in the QuickTime VR object.

The callback function must take a single parameter. In the callback function this parameter will be an object with two members:

nID – the ID of the hotspot that was clicked in a particular node.

nNodeID - the ID of the node the hotspot was clicked in.

Return true from the callback function if you want to cancel the normal handling of the specified hotspot; otherwise return false for default handling. For example, to prevent a URL node from opening a separate browser object, handle the node click yourself and return true from the callback function

Example:

Example 1:

To display hotspot information from a QuickTime VR object called qtvr without affecting it's functions:

```
qtvr.SetHotspotCallback(HSCallback)
function HSCallback(HSObject) {
      Debug.trace("Hotspot ID = (" + HSObject.nID + ", " + HSObject.nNodeID + ")\n")
      return false; // Allow the normal function of the hotspot }
```

Example 2:

To show an object called someObject when a specific hotspot is activated in a QuickTime VR object called qtvr:

```
qtvr.SetHotspotCallback(HSCallback)
function HSCallback(HSObject) {
      if ((HSObject.nID == 0) && (HSObject.nNodeID == 0))
{
someObject.Show() return true; // We handled it so prevent the hotspot doing anything else
} else {
            return false; // Allow the normal function of the hotspot
      } }
```

EnableHotspot

{bmc Pro_XE_04_Feature.bmp}

Syntax:

EnableHotspot(Type, ID, Enable)

Parameters:

Type – an integer specifying the type of nodes to be affected. This parameter is required.

Note:

The possible values for Type are:

Value of Type

Meaning

1 Enable/Disable hotspot with a given ID.

2 Enable/Diable hotspots of a particular type.

3 Enable/Disable all hotspots.

ID – an integer specifying the hotspot ID or hotspot type, depending on the Type value. This parameter is required.

Note: The possible values for ID depend on the setting of Type:

Value of Type

Possible values for ID

1 Hotspot ID

2 1 to Enable/Disable all Link hotspots 2 to Enable/Disable all URL hotspots 4 to Enable/Disable all Undefined hotspots

3 0

Enable – true to enable the hotspots, false to disable them. This parameter is required.

Remarks:

This function will enable or disable one, some or all of the hotspots in a QuickTime VR object.

Example 1:

To disable all hotspots in a QuickTime VR object called qtvr.

```
qtvr.EnableHotspot( 3, 0, false )
```

Example 2:

To enable all Link hotspots in a QuickTime VR object called qtvr.

```
qtvr.EnableHotspot( 2, 1, true )
```

Example 3:

To disable a specific hotspot in a QuickTime VR object called qtvr.

```
qtvr.EnableHotspot( 1, 21, false )
```

GetHotspotType

Syntax:

GetHotspotType(ID)

Return:

An integer representing the hotspot type. Possible values are:

Return Meaning

1 Link hotspot

2 URL hotspot

4 Undefined hotspot

Parameters:

ID – an integer specifying the hotspot ID. This parameter is required.

Remarks:

This function will return the type of a given hotspot in a QuickTime VR object.

Example:

To get the types of ten hotspots from a QuickTime VR object called qtvr.

```
for (hsID = 1; hsID <= 10; hsID++) {
HSType = qtvr.GetHotspotType( hsID ) if (HSType == 1) TypeName = " Link" else if (HSType == 2) TypeName = " URL" else if (HSType == 4) TypeName = "n Unknown" else TypeName = "n invalid" Debug.trace( "Hotspot " + hsID + " is a" + TypeName + " type (" + HSType + ")\n") }
```

Note:

This example assumes that there are 10 hotspots in the file and they are numbered from 1 to 10. This may not be the case for all QuickTime VR movies and will depend on the movie author.

SetHotspotType

Syntax:

SetHotspotType(ID, Type)

Parameters:

ID – an integer specifying the hotspot ID. This parameter is required.

Remarks:

This function is not currently implemented.

Example:

This function is not currently implemented.

GetVisibleHotspots

Syntax:

GetVisibleHotspots(ID)

Return:

An object array of visible hotspot IDs for the particular Node ID.

Parameters:

ID – an integer specifying the Node ID. This parameter is required.

Remarks:

This function will return an object array of visible hotspots for the particular Node ID.

Example:

To get a list of hotspots associated with a particular node from a QuickTime VR object called qtvr.

```
var NodeID = 2 var Hotspots = qtvr.GetVisibleHotspots( NodeID ) for ( Pos = 0; pos < Hotspots.length; pos++ ) {
      Debug.trace( "Hotspot id = " + Hotspots[Pos] + "\n" ) }
```

GetCurrentNode

Syntax:

GetCurrentNode()

Return:

An integer representing the current Node ID.

Remarks:

This function will get the current object Node ID of the QuickTimeVR object it is called on.

Example:

To get the current node from a QuickTime VR object called qtvr.

```
var CurrNodeID = qtvr.GetCurentNode()
```

SetMouseOverCallback

Syntax:

SetMouseOverCallback(MOHSCallback)

Parameters:

MOHSCallback – A function to call when the mouse cursor is over a hotspot in the QuickTime VR object. The function must take one parameter. This parameter is required.

Remarks:

This function will set a callback script function to be called when the mouse cursor is over a hotspot in the QuickTime VR object.

The callback function must take a single parameter. In the callback function this parameter will be an object with two members:

nID – the ID of the hotspot that the mouse is over in a particular node.

bEntering – True if the mouse has just entered the hotspot, false if it is exiting the hotspot.

Example:

To display respond to mouse over hotspot events from a QuickTime VR object called qtvr:

```
function MouseOverHSCallback(MOHSObject) {
iNode = qtvr.GetCurrentNode()
if (MOHSObject.bEntering) {
            Debug.trace("Entering hotspot " + MOHSObject.nID + " in node " +
iNode + "\n");
} else {
            Debug.trace("Leaving hotspot " + MOHSObject.nID + " in node " +
iNode + "\n");
      } }
qtvr.SetMouseOverCallback( MouseOverHSCallback )
```

Overview of Reference Lists

As an aide memoire, we have provided four quick reference lists of the OpusScript functions. Each list contains the function name, the category to which they belong and a one line description. You can click on the name of the function to open the full description of the function.

The OpusScript Reference Lists:

1. OpusScript Reference List – this lists all of the OpusScript functions that allow you to manipulate objects in your publication.

2. Database Reference List – this lists only the OpusScript functions

related to the Database feature in Opus.

3. Date Reference List – this lists only the Date functions, which are

the same as the ECMA-262 standard functions.

4. Math Reference List – this lists only the Math functions, which

are the same as the ECMA-262 standard functions.

ECMA-262 standard - definition

OpusScript is based on a JavaScript standard produced by the European Computer Manufacturers Association (ECMA), commonly known as ECMAScript or ECMA-262.

ECMA-262 is equivalent to JavaScript version 1.1. In other words, programming with OpusScript is like programming with JavaScript version 1.1 but obviously it has been extended with commands and functions specific to Opus publication objects.

OpusScript Reference List

This OpusScript Reference List is a list of all the OpusScript functions in alphabetical order. When using an OpusScript function in a Script Object or Script Action, you must enter the function name as it appears in this list.

Function Name: Category: Description:

Database A list of all the Database functions

Date A list of all the Date functions

A list of all the Math functions

Back Browsers Goes back one page in the browser history

Backward MultiFrames Go back one frame

bool Global String Convert a value to a Boolean

CaptureMouse Graphical

Objects

Capture the mouse input

ChangeDisplayMode Global Change the user's Display Mode

(i.e. screen resolution)

charAt String Get the character at the specified

position in the string

charCodeAt String Get the character code at the

specified position in the string

CharFromPoint Text Objects Get the character at a given position

ClearBookmark Pages Clear a Bookmark for the current page

CloneObject Graphical Objects Clone an object

Close File Close an opened file

Close Views Close the view

contains Global String Test if one string contains another string

Continue Clock Continue a paused clock Object

Continue Slideshow Continue a Slideshow

CopyFile Global Copy a file to a new location of the user's machine

CopyToClipboard Graphical Objects

Copy an object to the Windows clipboard

CopyToClipboard Pages Copy the page to the Windows clipboard

CreateCDPlayer Global Create a new CD player Object

CreateCDPlayer Sound Create a CD player Object

CreateClock Clock Creates a new clock Object

CreateHypertext Text Objects Create a hypertext link and action for a selection

Debug.trace Global Debugging tool that uses the Script Console

Debug.tracePane Global Debugging tool that creates a new pane in the Script Console

DeleteFile Global Delete a file from the user's machine

DeleteRegistryValue Global Delete a value from the system registry

DestroyClonedObject Graphical Objects Destroy a cloned object

Enable Graphical Objects Enable or disable this object

EndOfFile File Check for the end of file marker in the specified file

ExitPublication Global Exit the publication, via an Exit page (if any)

Fade Graphical Objects Fade an object by an amount

FileExists Global Test if a file exists in a givenlocation

FindChild Basic Objects Finds the child of this object with a particular name

FindDescendent Basic Objects Finds the descendent of this object with a particular name

FindText Text Objects Find a string of text in a Text object

FindTextInSelection Text Objects Find a string of text in a selection

FollowPath Graphical Objects

Make an object follow an animation path

fork Global Allows lines of code to be executed simultaneously

format Global String Format a number as a string

Forward Browsers Goes forward one page in the browser history

Forward MultiFrames Go forward one frame

GetAppearance Graphical Objects Get the appearance information for the Object State of a specified object

GetBookmarkPage Pages Get Bookmark page by its index number

GetChild Basic Objects Gets a child of this object by index

GetClock Clock Returns a specified clock Object

GetDisplayData Graphical Objects

Get display information related to an object e.g. scale and skew

GetFirstBookmark Pages Get the first Bookmark page

GetFirstChild Basic Objects Gets the first child of this object

GetFirstHypertext Text Objects Get the first hypertext link in a specified text object

GetFirstKeyword Search Gets first keyword in search list

GetFirstPage Search Gets first page to search on

GetFirstView Views Get the first view open

GetHeight Graphical Objects Get the height of the object

GetHours Clock Returns number of hours for a clock Object

GetINIFileData Global Return the value of a specified key in a specified section of an INI file

GetINISectionData Global Return the contents of a section of an INI file as a string

GetJoystickState Global Get the current joystick state

GetLastPage Global Get the last page in the publication

GetLastPage Publications Get the name of the last page in the publication

GetLayer Graphical Objects

Get the layer that the object is on

GetLength Video Get the length of the video in seconds

GetLineCount Text Objects Get the number of lines

GetMinutes Clock Returns number of minutes for a clock Object

GetMousePosition Pages Get the x and y coordinates of the current mouse position

GetName Basic Objects Gets the name of the object

GetNextBookmark Pages Get the next Bookmark page

GetNextChild Basic Objects Gets the child of this object that follows another

GetNextHypertext Text Objects Get the next hypertext link in a specified text object

GetNextKeyword Search Gets next keyword in search list

GetNextPage Pages Get the next page in the publication

GetNextPage Search Gets next page to search on

GetNextView Views Get the next view

GetNumberChildren Basic Objects Gets the number of children of this object

GetNumberHypertext Text Objects Get the total number of links in a specified text object

GetNumberViews Views Get the number of views open

GetObjectDimensions Graphical Objects Get the position and size of an object

GetPage Global Get a page by name

GetPage Publications Get a page by name

GetPage Views Get the page currently displayed in the view

GetPageDownloadPercent Global Get the percentage complete of the page downloaded

GetPageDownloadPosition Global Get the position of the page downloaded

GetPageDownloadTotal Global Get the total size of the page downloaded

GetPageNumber Pages Return page number of current page open in the publication

GetParagraphCount Text Objects Get the number of paragraphs in a Text object

GetParent Basic Objects Gets the parent of this object

GetPersistentObject Graphical Objects Get an object for storing information that will persist when the page is not visible

GetPosition Graphical Objects Get the x and y coordinate of the objects position

GetPosition Graphical Objects Get the x and y coordinate of an animation path’s length

GetPosition Sound Get the current position in seconds for a sound file

GetPosition Video Get the current position of a video in seconds

GetPositionFromPercent Graphical Objects Get the x and y coordinate of an animation path from a percent along the path

GetPreviousPage Pages Get the previous page in the publication

GetPublication Pages Get the name of the current publication

GetScrollInfo Graphical Objects Get the scroll position, scroll size and scrollbar

GetSeconds Clock Returns number of seconds for a clock Object

GetSelection Text Objects Gets the position of the selected characters

GetSelectionParagraphStyle Text Objects Get the paragraph style of selected text

GetSelectionStyle Text Objects Get the formatting style of selected text

GetSelectionText Text Objects Gets the selected text as a string

GetSlide Slideshow Get the slide currently visible

GetSlideCount Slideshow Count the number of slides in a slideshow

GetState Buttons Returns the current state of a push button

GetText Text Objects Get the text used for a hyperlink as a string

GetTextLength Text Objects Get the number of characters

GetTotalLength Graphical Objects Get the total length of an animation path

GetType Global Get the object's Object Type e.g. button, image etc.

GetType Views Get the View Type

GetUniqueObjectID Basic Objects Gets a string that is unique to this object

GetViewAt Views Get the view at a specified index position

GetVolume Sound Get the volume level of a specified device

GetVolume Sound Get the volume level of a specified sound Object

GetWidth Graphical Objects Get the width of the object

GetWordCount Text Objects Get the number of words

GetXPosition Graphical Objects Get the x coordinate of the objects position

GetYPosition Graphical Objects Get the y coordinate of the objects position

Go Video Set the first frame of a video

GotoBackwardPage Global Go to the previous page in the publication

GotoBookmark Pages Go to a specific Bookmark

GotoCurrentPage Global Go to current page in publication

GotoFirstLine File Goes to first line of an opened file

GotoForwardPage Global Go to the next page in the publication

GotoNextLine File Goes to next line of an opened file

GotoNextPage Global Go to the next page in the page history

GotoNextRandomPage Global Go to a random page within the
chapter without repeating
GotoPage Global Go to a given page
GotoPreviousPage Global Go to the previous page in the
page history
GotoRandomPage Global Go to a random page within the
chapter
GotoSlide Slideshow Go to a specific slide
Hide Graphical
Objects
Hide this object
Home Browsers Goes to the Home page of the
specified browser Object
indexOf String Return the position within the
String object of the substring
integer Global String Convert a value to an integer
InternetGetData Global Gets information from a remote server
InternetPostData Global Post information to a remote server
IsAutonarratePlaying Text Check if an autonarrate is currently playing
IsEnabled Graphical Objects Test if this object is enabled
IsKeyPressed Global Test if a specific key is being pressed
IsMousePressed Global Test if a specified special key is being pressed
IsObjectIntersecting Graphical Objects Test if this object is intersecting another
IsPlaying Slideshow Check if a specified Slideshow is playing
IsPlaying Video Check if a specified video is playing
IsShowing Graphical Objects Test if this object is showing
lastIndexOf String Return the position within the String object of the last match in the substring
LaunchFile Global Open an external file
LaunchSearch Global Open the Search dialog box
LaunchURL Global Open a URL
left Global String Get the left end of a string
length Global String Get the length of a string
length String The length of the string
LineFromChar Text Objects Get the line index of a character position
LineIndex Text Objects Get the character index of the first character in a line

LineLength Text Objects Get the number of characters in a line

MAX_VALUE Number The largest number that can be represented in OpusScript

mid Global String Get part of the middle of a string

MIN_VALUE Number The smallest number that can be represented in OpusScript

Move Graphical Objects Move the object by its x and y coordinates

MoveX Graphical Objects Move the object by its x coordinate

Move MoveY Graphical Objects the object by its y coordinate

NaN Number Returns a value that is "not a number"

Navigate Browsers Open a specified URL in specified browser Object

NEGATIVE_INFINITY Number A value representing negative infinity

number Global String Convert a value to a number

OpenFile File Opens a file and returns a new file Object

OpenSearch Global Returns a search object containing the publication search words

OpenSound Sound Open a sound file but don’t play it

ParagraphIndex Text Objects Returns the index number of the paragraph in a Text object

ParagraphLength Text Objects Returns the number of characters in a paragraph

Pause Clock Pause a clock Object

Pause Slideshow Pause a Slideshow

Play Slideshow Play a Slideshow

Play Sound Play a sound file

Play Video Play a specified video

PlayAutonarrate Text Objects Begin the autonarration on this text

PlayCDTrack Global Play a track from a CD

PlaySound Sound Play a sound file

PlaySystemSound Global Play a system sound from the users computer

PointFromChar Text Objects Get the position of a character

POSITIVE_INFINITY Number A value representing positive infinity

Print Browsers Print the contents of a Browser object

PrintFile Global Print the specified file

PrintObject Global Print the specified Object

PrintPage Global Print the specified page or page Object

random Global String Get a random integer

Read File Read a file that has been opened

ReadField File Reads a field in a file that has been opened

ReadLine File Reads first line of an opened file

ReadRegistryKey Global Return the value of a specified key in the Registry

ReallyExitPublication Global Exit the publication immediately

Refresh Browsers Redisplays the current page in the specified browser Object

RegisterEventHandler Event Objects Set a function to be called when a certain event triggers on the object.

ReleaseMouse Graphical Objects Release the mouse input

RemoveAllHypertext Text Objects Remove all hypertext links for this text

RemoveAlpha Graphical Objects Remove Alpha effect from specified object

RemoveBackground Graphical Objects Remove Background style from specified object

RemoveBorder Graphical Objects Remove Border style from specified object

RemoveBtnColour Graphical Objects Remove Button style from specified object

RemoveFlare Graphical Objects Remove Flare effect from specified object

RemoveImage Graphical

Objects

Remove Image file from specified object

RemoveShadow Graphical

Objects

Remove Shadow effect from specified object

RemoveTexture Graphical

Objects

Remove Texture effect from specified object

ReplaceSelection Text Objects Replaces the current selection with a specified string

ResetAnimation Graphical Objects Reset an animation

ResetVariables Publications Reset all the publications variables to their default

ResetVars Pages Reset all page properties variables

RGB Global Return a number indicating a RGB colour value

right Global String Get the right end of a string

Roll Graphical Objects Roll an object by a specified angle

Rotate Graphical Objects Rotate an object by a specified angle

Scale Graphical Objects Scale an object horizontally and vertically by a percentage

ScaleH Graphical Objects Scale an object horizontally by a percentage

ScaleV Graphical Objects Scale an object vertically by a percentage

Scroll Text Objects Scroll the text by line, paragraph or page

Seek Sound Reposition the starting point of a sound

Seek Video Reposition the starting point of a video

SendEmail Global Send an email

SetAlpha Graphical Objects

Set Alpha effect on a specified object

SetBackground Graphical

Objects

Set Background style on a specified object

SetBookmark Pages Bookmark the current page

SetBorder Graphical

Objects

Set Border style on a specified object

SetBtnColour Graphical

Objects

Set Button style on a specified object

SetColour Text Objects Set the colour of text in a Text
object
SetDisplayData Graphical
Objects
Set display information related to an object e.g.
SetFillColour Polygons Set the fill colour of a polygon in
a Vector object
SetFlare Graphical
Objects
Set Flare effect on a specified object
SetFocus Graphical
Objects
Set the keyboard input focus
SetImage Graphical
Objects
Set Image file on a specified object
SetLayer Graphical
Objects
Set the layer that the object is on
SetLineColour Polygons Set the line colour of a polygon in
a Vector object
SetListBoxSelection Text Objects Set the highlighted area in a
Listbox object to a new line
SetObjectDimensions Graphical
Objects
Set the position and size of an object
SetPosition Graphical
Objects
Set aspects of the objects position
SetPosition Sound Set the current position in seconds
for a sound file
SetPositionX Graphical
Objects
Set the x coordinate of the objects position
SetPositionY Graphical

Objects

Set the y coordinate of the objects position

SetRoll Graphical

Objects

Set an objects roll to a specified angle

SetRotation Graphical

Objects

Set an objects rotation to a specified angle

SetScale Graphical

Objects

Set the horizontal and vertical scale of an object to a percentage

SetScaleH Graphical

Objects

Set the horizontal scale of an object to a percentage

SetScaleV Graphical Set the vertical scale of an object

Objects to a percentage

SetScrollPosition Graphical

Objects

Set the scroll position of the object

SetSelection Text Objects Selects a range of characters in

the text object

SetSelectionParagraphStyle Text Objects Set the paragraph style of selected

text

SetSelectionStyle Text Objects Set the formatting style of

selected text

SetShadow Graphical

Objects

Set Shadow effect on a specified object

SetSkew Graphical

Objects

Skew an object horizontally and vertically to a specified angle

SetSkewH Graphical

Objects

Skew an object horizontally to a specified angle

SetSkewV Graphical

Objects

Skew an object vertically to a specified angle

SetSpin Graphical

Objects

Set an objects spin to a specified angle

SetState Buttons Sets the current state of a push

button

SetTexture Graphical

Objects

Set Texture effect on a specified object

SetTransparency Graphical

Objects

Set the objects transparency

SetVolume Sound Set the volume level for a

specified device

SetVolume Sound Set the volume level for a specified sound Object

Show Graphical Objects Show this object

ShowBookmarkDialog Pages Show the Bookmark Dialog box

Spin Graphical Objects Spin an object by a specified angle

split String Split a String object into an array of strings

Start Clock Start a clock Object

Start Timelines Start a specified Timeline
Stop Browsers Stops the specified browser
Object
Stop Clock Stop a clock Object
Stop Slideshow Stop a Slideshow
Stop Sound Stop a sound file
Stop Timelines Stop a specified Timeline
Stop Video Stop a specified video
StopAnimation Graphical
Objects
Stop an animation
StopAutonarrate Text Objects End the autonarration on this text
string Global String Convert a value to a string
substring String Returns a substring of a specified
string
TimeGetSeconds Publications Get the number of seconds a
publication has been running
ToEnd MultiFrames Go to last frame in the
MultiFrame
ToFrame MultiFrames Go to a given frame in the
MultiFrame
tolower Global String Get a lower case version of a
string
toLowerCase String Converts the String object to
lower case
ToStart MultiFrames Go to first frame in the
MultiFrame
toString Boolean Converts a Boolean object to a
string
toString File Returns the type of object as a
string
toString Number Converts the Number object into a
string
toString String Converts the String object into a
string

toupper Global String Get an upper case version of a

string

toUpperCase String Converts the String object to upper case

TriggerComponentEvent Event Objects Activate a Custom Trigger for a specified object.

UnregisterEventHandler Event Objects Remove a previously registered event handler.

valueOf Boolean Returns a primitive value of a Boolean Object

valueOf Number Returns the primitive value of the Number Object

valueOf String Returns the primitive value of the String object

word Global String Get a word from a string

Write File Write a string to an opened file

WriteField File Write a string to a field in an opened file

WriteLine File Write a string to a line in an opened file

WriteRegistryValue Global Write a value to the system registry

Database Reference List

The OpusScript Database functions are:

Function Name: Description:

AutoCommit Turn AutoCommit on or off for a database

Commit Commit to database – permanently stored on database

Connect Connect to SQL server

ExecuteSQL The SQL database command to execute

FirstRecord Go to First record in record set

GetCurrentRecordNumber Get current record number in record set

GetIdentQuoteChar Identify quote character required for table or field names containing spaces

GetLastError Get the last database error message

GetNumberOfRecords Get total number of records in record set

GetRecordAt Go to a specific record number in a record set

GetRecordAtRelative Go to a specific record number relative to current number

IsAutoCommitOn Check if AutoCommit is on in a database

LastRecord Go to Last record in record set

NextRecord Go to Next record in record set

PreviousRecord Go to Previous record in record set

Rollback Go back to point in database before changes were made or last commit sent

Date Reference List

The OpusScript Date functions are:

Function Name: Description:

Date.parse Returns the number of milliseconds since 1/1/1970

getDate Return the day of the month

getDay Return the day of the week

getFullYear Return the year

getHours Return the hour

getMilliseconds Return the number of milliseconds

getMinutes Return the minutes

getMonth Return the month

getSeconds Return the seconds

getTime Return the number of milliseconds since 1/1/1970

getTimezoneOffset Return the difference between local time zone and universal time in minutes

getUTCDate Return the day of the month in universal time

getUTCDay Return the day of the week in universal time

getUTCFullYear Return the year in universal time

getUTCHours Return the hour in universal time

getUTCMilliseconds Return the number of milliseconds in universal time

getUTCMinutes Return the minutes in universal time

getUTCMonth Return the month in universal time

getUTCSeconds Return the seconds in universal time

setDate Set the day of the month

setFullYear Sets the year

setHours Sets the hour

setMilliseconds Sets the number of milliseconds

setMinutes Sets the minutes

setMonth Sets the month

setSeconds Sets the seconds

setTime Sets the time since 1/1/1970 in milliseconds

setUTCDate Sets the date in the month in universal time

setUTCFullYear Sets the full year in universal time

setUTCHours Sets the hour in universal time

setUTCMilliseconds Sets the number of milliseconds in universal time

setUTCMinutes Sets the number of minutes in universal time

setUTCMonth Sets the month in universal time

setUTCSeconds Sets the seconds in universal time

toGMTString Converts the Date object to a string adjusted to GMT (Greenwich Mean Time)

toLocaleString Converts the Date object to a string adjusted to local time zone

toString Converts the Date object into a string

toUTCString Converts the Date object into a string adjusted to universal time

valueOf Returns the value of the Date object as a number of milliseconds

Math Reference List

The OpusScript Math functions are:

Function Name: Description:

Math.abs Returns the absolute value of a number

Math.acos Returns the arccosine of a number (in radians)

Math.asin Returns the arcsine of a number (in radians)

Math.atan Returns the arctangent of a number (in radians)

Math.atan2 Returns the angle from the X-axis to a point

Math.ceil Rounds a number up

Math.cos Returns the cosine of a number

Math.E Returns Euler's constant e (approx. 2.718)

Math.exp Returns E to the power of the argument passed

Math.floor Rounds a number down

Math.LN10 Returns the natural log of 10 (approx. 2.303)

Math.log Returns the natural log (base E) of a number

Math.LOG10E Returns the base 10 log of E (approx. 0.434)

Math.max Returns the larger of two values

Math.min Returns the smaller of two values

Math.PI Returns PI (approx. 3.1415)

Math.pow Returns the first parameter raised to the power of the second

Math.random Returns a pseudo-random number between 0 and 1

Math.round Rounds to the nearest integer

Math.sin Returns the sine of a number

Math.sqrt Returns the square root of a number

Math.SQRT1_2 Returns the square root of 1⁄2 (approx. 0.707)

Math.SQRT2 Returns the square root of 2 (approx. 1.414)

Math.tan Returns the tangent of a number

Using OpusScript with Flex & HTML5

Opus Flex (Adobe Flash) and HTML5 export supports a limited sub-set of the OpusScript functions. HTML5 export supports more than Flash and can be augmented with javascript.

It is strongly recommended that you begin with new scripts for Flash and HTML5 publications, as few existing scripts are likely to work without alteration. Please note, there are several limitations that are described below.

1. In Flash, all functions all return immediately. In OpusScript many functions do not return a value until the action they perform has completed (such as the Move() function). It is not possible to replicate this behaviour in Flex, therefore all functions return immediately.

2. Variables in Flash are case insensitive. For example, if you have a variable named "Button" and another called "BUTTON" then they will refer to the same thing when published in Flex. During the publish process, Flex warns about situations where this may be occurring the Publishing Details. These warnings may occur when there is no problem, if for example two variables are local variables in separate functions.

3. Many OpusScript functions are not implemented in Flash, in the following chapter is a list of OpusScript functions which are supported. Please check the Help file provided with the program to see which features are not supported in HTML5 output.

4. The for and if loops do not work in Flash. However, as with many of the other unsupported functions most things that can be accomplished in OpusScript can be accomplished but may require a different approach.

Flash ActionScripts

As well as the OpusScript functions, Opus Flex does support some of the Macromedia Flash ActionScripts. The ActionScripts can be included in a Script Object or Script Action, however, you will not see the result of these actions in Opus when you preview the page or publication. To see the ActionScript in operation, you will have to publish your publication and then run the published version either in a browser or the Macromedia Flash Player.

Please note, not all of the ActionScripts are available within Flex and there is a potential that some ActionScript names could conflict with names you have used within your OpusScript. You will therefore need to establish this by testing out what you are trying to do in a simple initial prototype.

OpusScript functions supported by Flex

This OpusScript Reference List is a list of all the OpusScript functions in alphabetical order that are supported by Opus Flex. When using an OpusScript function in a Script Object or Script Action, you must enter the function name as it appears in this list.

Function Name: Category: Description:

Date All the Date functions

Math All the Maths functions

Backward MultiFrames Go back one frame

bool Global String Convert a value to a Boolean

ClearBookmark Pages Clear a Bookmark for the current page

contains Global String Test if one string contains another string

Continue Slideshow Continue a Slideshow

Enable Graphical Objects Enable or disable this object

ExitPublication Global Exit the publication, via an Exit page (if any)

Fade Graphical Objects Fade an object by an amount

FindChild Basic Objects Finds the child of this object with a particular name

FindDescendent Basic Objects Finds the descendent of this object with a particular name

format Global String Format a number as a string

Forward Browsers Goes forward one page in the browser history

Forward MultiFrames Go forward one frame

GetBookmarkPage Pages Get Bookmark page by its index number

GetChild Basic Objects Gets a child of this object by index

GetFirstBookmark Pages Get the first Bookmark page

GetFirstChild Basic Objects Gets the first child of this object

GetHeight Graphical Objects Get the height of the object

GetLayer Graphical Objects Get the layer that the object is on

GetLength Video Get the length of the video in seconds

GetMousePosition Pages Get the x and y coordinates of the current mouse position

GetName Basic Objects Gets the name of the object

GetNextBookmark Pages Get the next Bookmark page

GetNextChild Basic Objects Gets the child of this object that follows another

GetNumberChildren Basic Objects Gets the number of children of this object

GetParent Basic Objects Gets the parent of this object

GetPersistentObject Graphical Objects Get an object for storing information that will persist when the page is not visible

GetPosition Graphical Objects Get the x and y coordinate of the objects position

GetPosition Video Get the current position of a video in seconds

GetSlide Slideshow Get the slide currently visible

GetSlideCount Slideshow Count the number of slides in a slideshow

GetState Buttons Returns the current state of a push button

GetType Global Get the object's Object Type e.g. button, image etc.

GetUniqueObjectID Basic Objects Gets a string that is unique to this object

GetVolume Sound Get the volume level of a specified device

Get GetWidth Graphical Objects the width of the object

GetXPosition Graphical Objects Get the x coordinate of the objects position

GetYPosition Graphical Objects Get the y coordinate of the objects position

Go Video Set the first frame of a video

GotoBackwardPage Global Go to the previous page in the publication

GotoBookmark Pages Go to a specific Bookmark

GotoForwardPage Global Go to the next page in the publication

GotoNextPage Global Go to the next page in the page history

GotoNextRandomPage Global Go to a random page within the chapter without repeating

GotoPreviousPage Global Go to the previous page in the page history

GotoRandomPage Global Go to a random page within the chapter

GotoSlide Slideshow Go to a specific slide

Hide Graphical Objects Hide this object

integer Global String Convert a value to an integer

IsEnabled Graphical Objects Test if this object is enabled

IsObjectIntersecting Graphical Objects Test if this object is intersecting another

IsPlaying Slideshow Check if a specified Slideshow isplaying

IsPlaying Video Check if a specified video is playing

IsShowing Graphical Objects Test if this object is showing

LaunchSearch Global Open the Search dialog box

LaunchURL Global Open a URL

left Global String Get the left end of a string

length Global String Get the length of a string

length String The length of the string

mid Global String Get part of the middle of a string

Move Graphical Objects Move the object by its x and y coordinates

MoveX Graphical Objects Move the object by its x coordinate

MoveY Graphical Objects Move the object by its y coordinate

number Global String Convert a value to a number

Pause Slideshow Pause a Slideshow

Play Slideshow Play a Slideshow

Play Video Play a specified video

random Global String Get a random integer

ReallyExitPublication Global Exit the publication immediately

RegisterEventHandler Event Objects Set a function to be called when a certain event triggers on the object.

ResetAnimation Graphical Objects Reset an animation

ResetVariables Publications Reset all the publications variables to their default

ResetVars Pages Reset all page properties variables

right Global String Get the right end of a string

Rotate Graphical Objects Rotate an object by a specified angle

Scale Graphical Objects Scale an object horizontally and vertically by a percentage

ScaleH Graphical Objects Scale an object horizontally by a percentage

ScaleV Graphical Objects Scale an object vertically by a percentage

Seek Video Reposition the starting point of a video

SetBookmark Pages Bookmark the current page

Set SetLayer Graphical Objects set the layer that the object is on

SetPosition Graphical Objects Set aspects of the objects position

SetPosition Sound Set the current position in seconds for a sound file

SetPositionX Graphical

Objects

Set the x coordinate of the objects position

SetPositionY Graphical

Objects

Set the y coordinate of the objects position

SetRotation Graphical

Objects

Set an objects rotation to a specified angle

SetScale Graphical

Objects

Set the horizontal and vertical scale of an object to a percentage

SetScaleH Graphical

Objects

Set the horizontal scale of an object to a percentage

SetScaleV Graphical

Objects

Set the vertical scale of an object to a percentage

SetSelection Text Objects Selects a range of characters in the

text object

SetState Buttons Sets the current state of a push button

SetTransparency Graphical

Objects

Set the objects transparency

Show Graphical

Objects

Show this object

ShowBookmarkDialog Pages Show the Bookmark Dialog box

Stop Slideshow Stop a Slideshow

Stop Video Stop a specified video

StopAnimation Graphical

Objects

Stop an animation

string Global String Convert a value to a string

TimeGetSeconds Publications Get the number of seconds a publication has been running

ToEnd MultiFrames Go to last frame in the MultiFrame

ToFrame MultiFrames Go to a given frame in the

MultiFrame

tolower Global String Get a lower case version of a string

ToStart MultiFrames Go to first frame in the MultiFrame

toupper Global String Get an upper case version of a string

TriggerComponentEvent Event Objects Activate a Custom Trigger for a specified object.

UnregisterEventHandler Event Objects Remove a previously registered event handler.

word Global String Get a word from a string