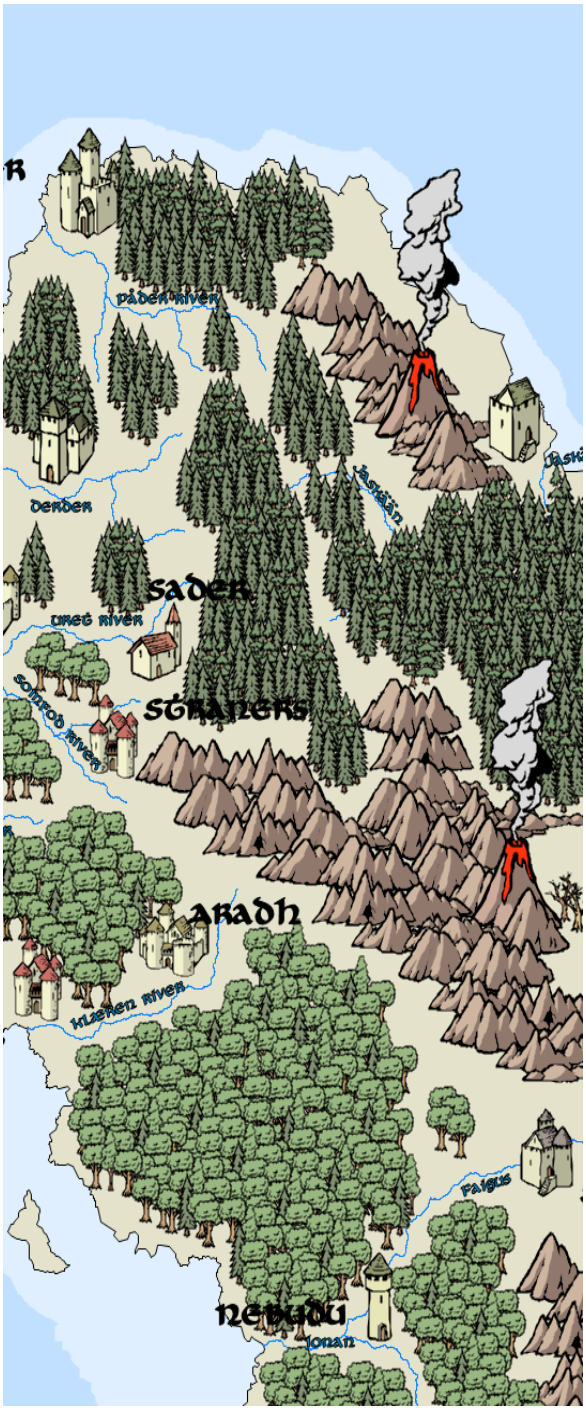

Fractal Mapper SDK & GoblinAPI

Document Date 2/14/2008

Table of Contents

Part III GoblinAPI Scripting Interface 9

1 Plugins

1.1 Command Tags

When Fractal Mapper starts, it examines the plug-in files located in the plug-ins directory. It opens each plug-in file, and checks the command tags located within it. These tags tell the program what type plug-in the script represents and various bits of data about the plug-in.

Command tags are simply formatted lines placed into the script. They are preceded with '#' symbols. Here's some examples:

```
#plugin         Quick Generate
#author         NBOS
#desc           Generates random systems until the Escape key is pressed
#button         ExampleStuff\QuickGenerate.bmp
```

There are several different types of plug-ins supported by Fractal Mapper. Scripts need to have command tags telling the program that the script is one of the available types, or the plug-in won't be loaded. The plug-in types are:

Program Feature. These plug-ins add menu items to the program, essentially creating a new feature in the program. Feature plug-ins are defined with the #plugin 5 tag.

Pop-up Menu Feature. These plug-ins add an item to the pop-up menu that is displayed when the user right-clicks on the map. These are defined with the #popmenu 5 tag.

File Handler. These plug-ins allow script creators to enable the program to respond to or import certain types of files. These plug-ins are defined with the #filehandler 3 tag.

Custom Tools. These are scripts that add a custom drawing tool to the Custom Tools palette. These plug-ins are defined with the #customtool 2 tag.

Inspiration Pad Generator. These plug-ins allow script creators to hook into the built-in Inspiration Pad and add to the list of available generators. These plug-ins are defined with the #ipad 3 tag.

1.2 #author

```
#author name
```

The #author tag lets you tell the program who created the script. name is your name, and is displayed in the Plug-in manager when the plug-in is selected.

1.3 #button

```
#button file
```

The #button tag lets you define a toolbar button image for a Program Feature plug-in. *File* is a bitmap file 16 pixels high, and up to 64 pixels wide. The path should be relative to the program's 'Plugins' directory.

Example:

```
#button MyToolbarImage.bmp
```

1.4 #customtool

```
#customtool name
```

The #customtool tag tells the program that this plug-in is a custom drawing tool. *Name* is the name of the drawing tool, and will be displayed as the tool tip for the tool when the mouse is hovered over it. The #toolbase and the #button tag are also used in conjunction with #customtool.

Example:

```
#customtool     Dungeon Room
#toolbase       Rectangle
#button         Dungeon Room.bmp
```

1.5 #desc

```
#desc text
```

The #desc tab defines a short descriptive text that is displayed in the plug-in management tab when the plug-in is selected.

Example:

```
#desc     Sample tool to create a dungeon room
```

1.6 #desc_

```
#desc_language text
```

The #desc_ tab allows script creators to assign script descriptions in languages other than English. *Language* is the name of the language, and *text* is the text description of the plug-in in that language. This description will be shown when the program is running using the specified language. Any number of language entries can be used, so long as each is a different language.

Example:

```
#desc_nederlands Fractale Cirkel
```

1.7 #extdesc

```
#extdesc text
```

The #extdesc tag defines an extended description of the plug-in. This information is displayed at the bottom of Plug-in management tab when the plug-in is selected in the plug-in management page.

1.8 #fileext

```
#fileext file-extension
```

The #fileext tag defines the file extension being handled by a file handler plug-in. The '.' in the file extension should be included.

Example:

```
#filehandler
#fileext  .sgs
```

This tells the program that this plug-in is used to handle files with a '.sgs' extension.

1.9 #filehandler

```
#filehandler plugin-name
```

The #filehandler tag tells the program that this plug-in is a File Handler plug-in. *Plugin-name* is the name of the plug-in. This tag is used along with #fileext.

Example

```
#filehandler PNG importer
#fileext .png
```

1.10 #ipad

```
#ipad name
```

The #ipad tag tells the program this is an Inspiration Pad plug-in. These plug-ins can be used to hook into the built-in Inspiration Pad to add new generators. *Name* is the name of the plug-in, and is what is displayed in the Inspiration Pad.

Example:

```
#ipad     Fantasy Names
```

1.11 #menuhotkey

```
#menuhotkey key-combo
```

This tag defines the menu hot key for program feature plug-ins that add to the program menu. *Key-combo* is a valid Windows hot-key combination.

Examples:

```
#menuhotkey Shift+F5
#menuhotkey F5
#menuhotkey Ctrl+B
#menuhotkey Alt+F1
#menuhotkey Alt+W
#menuhotkey Shift+F12
```

1.12 #menuinsert

```
#menuinsert location
```

For program feature plug-ins, this lets you specify where to put the menu item for this plug-in. By default all plug-ins are placed on the Plugins menu. By setting values for #menuinsert, you can change this. The positions are defined by the menu number, from left to right. That is, the File menu is 1, Edit is 2, etc. If you use a single number, the plug-in is added to the bottom of that menu's list of items. If you use two numbers separated by a space in the #menuinsert tag, the plug-in is placed at the position of the second number within the menu defined by the first number.

Examples

places the plug-in as the 11th entry of the first menu. In a default installation, this is right after the Upload entry in the File menu.

```
#menuinsert 1 11
```

places the plug-in at the end of the 5th menu, typically the Actions menu.

```
#menuinsert 5
```

Note that menu divider lines are counted as menu items.

1.13 #name_

```
#name_language name
```

The #name_ tag allows the script creator to assign alternate names to the plug-in so they can display their names in different languages. *Language* is the language name, and *name* is the name of the plug-in which will display when the program is set to use *Language*. Any number of #name_ tags can be used, as long as each is a different language.

Example:

```
#name_nederlands Fractale Cirkel
```

1.14 #plugin

```
#plugin name
```

This tells the program the script is a Program Feature plug-in. These plug-ins create menu items that can be used to call the plug-in. *Name* is the name of the plug-in, and is what is displayed on the menu.

Example:

```
#plugin Rotate Objects 45 deg.
```

1.15 #popmenu

```
#popmenu name
```

This tag tells the program the plug-in is a Pop Up Menu plug-in. These plug-ins are made available to run as items in the pop-up menu displayed when you right-click on the map. *Name* is the name of the plug-in, and is what is displayed on the menu. This tag is often used in conjunction with the #button and #menuhotkey tags.

Example:

```
#popmenu  Flatten
#button         flatten.bmp
#menuhotkey     Shift+F1
```

1.16 #template

not yet implemented

1.17 #toolbase

```
#toolbase tool-name
```

The #toolbase tag defines the base drawing tool type for a custom tool. Each custom tool must have a base tool. This base type is one of the standard drawing tools. When the custom tool is used, it operates exactly as the base type tool operates until the drawing action is complete - then the script takes over and performs its tasks.

The available base tool types are:

```
Land Mass
Text
Polygon
Polyline
Triangle
Line
PolyPath
Freehand Line
Freehand Polygon
Hallway
Spline Curve
Closed Spline
Symbol
```

```
Symbol Fill
Smart Building
Rectangle
Straight Line
Circle
Angled Rectangle
Triangle Land Mass
```

Use the name of the base tool *exactly* as listed above.

Example

```
#customtool     Fractal Circle
#toolbase       Circle
#button         fractalcircle.bmp
```

1.18 #toolset

not yet implemented

2 Event Handling

2.1 Responding to Events

The GoblinAPI allows you assign a subroutine to run when certain 'events' occur within the program. For example, you can tell the program to run a certain subroutine when the map is clicked on with the pointer tool. To have the program execute a subroutine to respond to certain events, create a subroutine of the same name (called an 'event handler') within the map's script module (Plugins-Edit Map Scripts from the menu). For example, if you want certain code to execute when the map is zoomed, create a subroutine called 'OnZoom()'.

Here's an example of responding to the OnLoad Event. In this subroutine, a Microsoft Agent is loaded ('Merlin', in this case), who then introduces himself.

```
dim Agent, Char, Request
AgentName = "Merlin"
sub OnLoad()
   set Agent = CreateObject( "Agent.Control.2")
   Agent.Connected = true
   Agent.Characters.Load AgentName, AgentName + ".acs"
   set Char = Agent.Characters( AgentName)
   if Char.Visible = false then
      Char.Show
   end if
   Char.Play "Greet"
   Char.Play "RestPose"
   Char.Speak "Well hello there!  Welcome to my favorite town!"
   Char.Speak "Just click on any of the buildings with the pointer tool..."
   Char.Speak "and I'll tell you all about it!"
   SetGlobalVariant 1, Agent
end sub
```

The list of available events follows.

2.2 OnAdd

```
OnAdd()
```

This event is fired whenever an object is added to the map manually (not by a script). This event fires after any Custom Tool scripts.

Example:

```
'increments the text of a map object (the first
'object on the map) each time an object is added
sub OnAdd
   Dim o
   On Error Resume Next
   o = GetMapObject( 1)
   if MapperErrorCode = 0 then
      o.Text = GetMapObjectCount()
   end if
end sub
```

2.3 OnConfigure

```
OnConfigure()
```

This event is called when selected objects are altered.

2.4 OnDelete

```
OnDelete()
```

This event is called when an object is deleted from the map manually. This does not apply to deletions performed by plugins.

2.5 OnLoad

```
OnLoad()
```

This event is triggered when the map is loaded from disk. You can use this to do things like extract embedded table files, or run some randomizing routines on the map (making a dungeon map that's different each time you load it, for example).

2.6 OnMove

```
OnMove()
```

This event is fired whenever an object on the map is moved.

2.7 OnQuit

```
OnQuit()
```

This event is called when the program is closed.

2.8 OnSave

```
OnSave()
```

This event is triggered whenever the map is saved. You can use this event to do things like embed necessary table data files or clear out unneeded map objects.

2.9 OnScroll

```
OnScroll()
```

This event is called whenever the map is scrolled using the scroll bars.

2.10 OnSelect

```
OnSelect()
```

This event fires whenever the user clicks on the map with the pointer tool. Map object selection processing is performed first by the program, so that objects referenced by the event script have their proper selection state in response to the click.

2.11 OnTextEdit

```
OnTextEdit()
```

This event is called when the text of a text object is changed by the user. This is useful when you need to respond to the value of a text object being changed. For example, you can make a character sheet that automatically updates secondary information from a table when an attribute is changed.

2.12 OnTimer

```
OnTimer()
```

If the timer has been set using the SetTimer() 22 function, this event fires at intervals specified by the

parameter passed to the SetTimer() function. You can use this event to animate objects on the map.

2.13 OnUnload

```
OnUnload()
```

This event is triggered when the map is unloaded (by closing the program, selecting File-New from the menu, or loading another map).

2.14 OnZoom

```
OnZoom()
```

This event is triggered when the zoom factor of the map changes.

3 GoblinAPI Scripting Interface

3.1 Pre-Processor Commands

3.1.1 #append

```
#append {file}
```

Tells the preprocessor to append the contents of the specified file to the end of the current script. This is very useful when you've created a set of functions and subroutines that you wish to use within other scripts. Once a file has been appended, you can call it's functions within your script.

Example:

```
'adds the contents of the file called 'randomnames.FMScript'
'to the current script.
#append randomnames.FMScript
```

3.1.2 #insert

```
#insert {file}
```

Tells the preprocess to insert the specified file at the current location of your script. This differs from the #append directive, which always adds the specified file to the end of the script. Once a file has been inserted, it's functions and subroutines are available.

Example:

```
'inserts the contents of 'mytables.FMScript' at this line:
#insert mytables.FMScript
```

3.2 General API Functions

3.2.1 Base64DecodeToFile

```
Base64DecodeToFile( sData, sFile: string)
```

Decodes the base64 data stored in sData, and saves the result to the file named sFile. Use this function to extract data encoded with the Base64EncodeFile() function.

Example: (see the Base64EncodeFile() example)

3.2.2 Base64EncodeFile

```
Base64EncodeFile( sFile: string): string
```

Returns the contents of file sFile as a base 64 encoded string (similar to the way email programs attach binary files). This string is suitable for use as a custom field value (thus enabling you to embed files within your maps).

Example:

```
Sub AttachFile()
   Dim s
   s = Base64EncodeFile( "c:\photo.jpg")
   SetMapField "photo", s
End Sub

Sub ExtractFile()
   Dim s
   s = GetMapField( "photo")
   If Len( s) > 0 Then
      Base64DecodeToFile s, "c:\photo2.jpg"
   End If
End Sub
```

3.2.3 DebugClear

```
DebugClear()
```

Clears the contents of the debugging window.

3.2.4 DebugMsg

```
DebugMsg( s: string)
```

Outputs the string s to the debugging window. If the debugging window has not already displayed by by previous calls to this function, the window is displayed.

3.2.5 FileHandlerFileName

```
FileHandlerFileName(): string
```

Used by File Handler type plugins. Returns the name of the file that was selected when the user tries to open a file of a registered file type.

3.2.6 FreeObject

```
FreeObject( o: object)
```

Frees an object that is dynamically created within a script. Objects created with CreateMapObject(), NewMap(), and NewIPadTable() should be deleted with this function when they are no longer needed. Don't use this function on objects you want to keep - for example, a map object you've added to your map.

This function does not apply to COM objects created with Window's own CreateObject() VBScript function.

3.2.7 GenerateGroupID

```
GenerateGroupID(): integer
```

Returns an integer that can be used as a group id. This id can be assigned to one or more MapObject's Group property to group them together.

3.2.8 GetAppDataDir

```
GetAppDataDir(): string
```

Returns the path to Fractal Mapper's Application Data directory. This is the directory in which Fractal Mapper stores it's internal data files and temporary files.

Typically the directory will look something like (under Windows XP):

```
C:\Documents and Settings\<User Name>\Application Data\NBOS\Fractal Mapper
```

3.2.9 GetCurrentMapDir

```
GetCurrentMapDir(): string
```

This returns the name of the directory in which the currently opened map file resides.

3.2.10 GetCurrentMapFile

```
GetCurrentMapFile(): string
```

Returns the name of the map file currently displayed by the program.

3.2.11 GetGlobalVariant

```
GetGlobalVariant(n: integer): olevariant
```

Retrieves the OLE object stored at position n in the global OLE object array.

Example: (See the SetGlobalVariant() example)

3.2.12 GetMapperDir

```
GetMapperDir(): string
```

Returns the directory that the Fractal Mapper executable is running from, including trailing backslash. This will typically look something like

```
c:\Program Files\nbos\mapper8\
```

3.2.13 GetMapperScriptsDir

```
GetMapperScriptsDir(): string
```

This is identical to GetPluginDir() 12.

3.2.14 GetPluginDir

GetPluginDir(): string

Returns the name of the program's plugin directory. Typically, this would look something like

```
c:\Program Files\nbos\Mapper8\Plugins\
```

3.2.15 MapperApplication

```
MapperApplication(): integer
```

Reports the application that is running the script. For Fractal Mapper, this always returns 1.

Should the need arise in the future, other applications may report this value differently, allowing scripts to adjust according to their host application.

3.2.16 MapperBuild

```
MapperBuild(): integer
```

Returns an integer representation of the running version of Fractal Mapper, including the major and minor version identifier. For example, Fractal Mapper 8.01 would return a value of 801. As new versions become available, this function will be helpful in making sure the script is executing on the required version of Fractal Mapper.

3.2.17 MapperClearError

```
MapperClearError()
```

Clears the last mapper error code and error message. After this is called, MapperErrorCode will return 0 until another error is raised by the mapper.

3.2.18 MapperErrorCode

```
MapperErrorCode(): integer
```

Returns the last internal error code raised by the mapper, or zero if no error. You can check this after attempting to access internal objects via GoblinAPI.

Example:

```
Dim o
o = GetMapObject( 10)
If MapperErrorCode = 0 Then
        ...do things With o...
   Else
   MsgBox MapperErrorCode
End If
```

You can also use NotNull() 14 to check for valid objects.

3.2.19 MapperErrorText

```
MapperErrorText(): string
```

Returns the last error message generated by the mapper, or an empty string if no error message

exists. If an error code is reported by MapperErrorCode(), you can use the MapperErrorText() function to return the text of the error.

Example: (see the MapperErrorCode() example)

3.2.20 MapperVersion

```
MapperVersion(): integer
```

Identical to MapperBuild() 13

3.2.21 NotNull

```
NotNull( o: object): boolean
```

Returns true if the object o is not null. This function is useful error checking objects created with CreateMapObject(), NewMap(), and NewIPadTable().

Example:

```
'Checks that an object was successfully created
o = CreateMapObject( "Polygn")  'note the typo
If NotNull( o) Then
   o.BrushColor = ....
   ...
Else
   MsgBox "Could not create the object"
End If
```

Example #2

```
'Checks that a requested object actually exists
oMap = GetCurrentMap()
o = oMap.GetObject( 4021)
If NotNull( o) Then
   MsgBox "This object Exists"
End If
```

3.2.22 RSeed

```
RSeed()
```

Re-seeds the program's internal random number generator. In most cases this is handled automatically by the program, but there may be cases where a new random number seed is desired to prevent repeating patterns.

Note that this has no effect on the VBScript random number generator, which is an entirely different random number generation system.

3.2.23 SetGlobalVariant

```
SetGlobalVariant( n: integer, v: olevariant)
```

Sets one of 20 allocated entries in an array of OLE objects to v. You can use the SetGlobalVariant() and GetGlobalVariant() to maintain a an OLE session in between script calls. For example, you can create a Web Browser or MS-Agent object in the OnLoad() event, and then maintain the connection to that object for use during various events.

Example:

```
Sub OnLoad()
   Set Agent = CreateObject( "Agent.Control.2")
   ... Do things With the agent object...
   SetGlobalVariant 1, Agent
End Sub

Sub OnSelect()
   Dim o, n, s
   n = GetSelectedIndex()

   ' check if we've clicked on an object
   If n > 0 Then
      Set Agent = GetGlobalVariant(1)
      ...do things With the Agent object...
   End If
End Sub
```

3.2.24 ShowMapStats

```
ShowMapStats()
```

Calculates and displays the current map's statistics. Includes the object count, node count, and rendered node count with and without submap objects.

3.3 Dice Functions

3.3.1 d2

```
d2(): integer
```

Rolls a single d2 (i.e., as if a dice with 2 sides) and returns the result.

3.3.2 d4

```
d4(): integer
```

Rolls a single d4 and returns the result.

3.3.3 d6

```
d6(): integer
```

Rolls a single d6 and returns the result.

Example:

```
'Rolls for 'wandering damage' - one of each type.
n = d2() + d4() + d6() + d8() + d10() + d12() + d20() _
      + d30() + d100()
```

3.3.4 d8

```
d8(): integer
```

Rolls a single d8 and returns the result.

3.3.5 d10

```
d10(): integer
```

Rolls a single d10 and returns the result.

3.3.6 d12

```
d12(): integer
```

Rolls a single d12 and returns the result.

3.3.7 d20

```
d20(): integer
```

Rolls a single d20 and returns the result.

3.3.8 d30

```
d30(): integer
```

Rolls a single d30 and returns the result.

3.3.9 d100

```
d100(): integer
```

Rolls a d100 (a percentile, 1 to 100) returns the result.

3.3.10 EvalDice

```
EvalDice( sDice: string): integer
```

Evaluates a dice expression expressed in a string of the form {n}d{t}+/-{m}, where n is the number of dice, t is the die type, and m is a modifier to add or subtract.

Example:

```
'Rolls 5d6-3
n = EvalDice( "5d6-3")
```

3.3.11 RollDice

```
RollDice( nNumDice: integer, nDieType: integer, nModifier: integer): integer
```

Generates a dice roll of the expression {nNumDice}d{nDieType}+{nModifier}.

Example:

```
'rolls 3d6+2
n = RollDice( 3, 6, 2)
```

3.4 Translation Functions

3.4.1 GetCurrentLanguage

```
GetCurrentLanguage(): string
```

Returns the language to which the program's GUI is currently set.

3.4.2 FormatMessage

```
FormatMessage( s: string; p1, p2, p3, p4, p5, p6: string): string
```

Translates the message *s*, applying the parameters *p1 - p6,* using the currently set language data files.

Some messages in the language data files have place holders for parameters. Use this method to access those types of messages. This does not perform a translation - it just accesses the language data files for a translated message of *s*.

Example:

```
s = FormatMessage( "Grid Size in %1", "Miles", "", "", "", "", "")
MsgBox s
```

3.4.3 TranslateMessage

```
TranslateMessage( s: string): string
```

Returns a translation for *s* in the currently set language, or *s* if the translated message is not available in the translation data files. This does not actually perform a translation - it just looks up the message in the data files.

3.5 Inspiration Pad Functions

These are functions that related to adding and removing items from the built-in Inspiration Pad. This is not to be confused with functions relating to the creation and use of Inspiration Pad objects.

3.5.1 InspirationPadAdd

```
InspirationPadAdd(sLine: string)
```

This function adds *sLine* to the list of generated names on the Inspiration Pad.

3.5.2 InspirationPadClear

```
InspirationPadClear()
```

This function clears all the existing generated items in the Inspiration Pad. Use this at the beginning of each Inspiration Pad script.

3.5.3 InspirationPadShow

```
InspirationPadShow()
```

Use this function to display the Inspiration Pad. While not strictly necessary, this allows your script to open and display the Inspiration Pad if it is called from one of the Inspiration Pad quick access menu items.

Example:

```
'Inspiration Pad Example
#Append nbos\randomwords.vbs

Sub Main
   Dim i

   InspirationPadClear  'clear existing entries
   For i = 1 To 100
      'generate a random name and add it to the Inspiration Pad
      InspirationPadAdd( RandomTownName( "Albanian"))
   Next
   InspirationPadShow
End Sub
```

3.6 Program Interface Functions

A note on screen and world coordinates. Each map has an internal coordinate system that ranges from (0,0) to (10000000, 10000000), with (0,0) being the top left part of the map, and (10000000, 10000000) being the bottom right part of the map. These are referred to as *'world'* coordinates by the GoblinAPI. The points (nodes) of each MapObject object are stored in world coordinates. When it comes time to draw the object on the screen, the mapper's graphics engine will convert the 'world' coordinates of each object to where they should be represented on the screen. The converted values are the 'screen' or 'pixel' coordinates.

For example, the world point (10000000, 10000000) represents the bottom right of the map. When the map is displayed at a zoom level of 4% (and without any horizontal or vertical scrolling), this point is represented by the pixel location (400,400) on the map display on your screen.

3.6.1 CenterMapOnXY

```
CenterMapOnXY( x:integer, y: integer)
```

Centers the map on the point (x, y), where x is the horizontal measurement in logical map units, and y is the vertical measurement. Point (0,0) represents the top left corner of the map, and (10000000, 10000000) represents the bottom right (each map has a precision of 10,000,000 logical units across). The (x,y) combination do not represent the values of the map span units (feet, miles, etc). Each map, regardless of the span in the chosen unit has a precision of 10,000,000 units. When a map is not square, it's largest side is set to 10,000,000, and the smaller one set to the appropriate proportion of 10,000,000.

Example:

```
'prompts the user for the name of a city, and
'cycles through all the objects looking for it
'by comparing the object's text value.  If found,
'it zooms in and centers in on the city
Sub Main()
   Dim s, n, i, o, x, y
   Dim isFound

   isFound = False
   s = InputBox( "Name of the city to find")
   If Len( s) > 0 Then
      n = GetMapObjectCount()
      For i = 1 To n
         o = GetMapObject( i)
         If UCase(o.Text) = UCase(s) Then
            x = o.GetPointX( 1)
            y = o.GetPointY( 1)
            SetMapZoom 45
            CenterMapOnXY x, y
            isFound = True
            Exit For
         End If
      Next

      If isFound = False Then
         MsgBox "City isn't found"
      End If
```

```
      End If
   End Sub
```

3.6.2 DisablePatterns

```
DisablePatterns()
```

Disables the display of Pattern Fills on your map.

3.6.3 EnablePatterns

```
EnablePatterns()
```

Enables the display of Pattern Fills on your map.

3.6.4 GetKey

```
GetKey(): Integer
```

Returns an integer representing the first key found in the message queue, or -1 if no key has been pressed. Use this function to check for the Escape key being pressed (27), or even control navigation by responding to arrow key presses.

Example:

```
#plugin GetKey Test
'loops until the Escape Key (#27) is pressed
Do While True
    k = GetKey()
    If k = 27 Then Exit Do

    If k > 0 Then
          MsgBox "You pressed keycode: " & k & ".  To stop this script click OK and
    End If
Loop

MsgBox "Script Stopped!"
```

3.6.5 GetMouseX

```
GetMouseX()
```

Returns the last known x screen coordinates (in pixels) of the mouse over the map display area. Use ScreenToWorldX() to convert these values to their corresponding 'world' values.

3.6.6 GetMouseY

```
GetMouseY()
```

Returns the last known y screen coordinates (in pixels) of the mouse over the map display area. Use ScreenToWorldY() to convert these values to their corresponding 'world' values.

3.6.7 GetScreenHeight

```
GetScreenHeight()
```

Returns the height of the map display area of the program, in pixels.

3.6.8 GetScreenWidth

```
GetScreenWidth()
```

Returns the width of the map display area of the program, in pixels.

3.6.9 GetStatusMsg

```
GetStatusMsg(): string
```

Returns the current message displayed in the program's status bar.

3.6.10 Pause

```
Pause( n: integer)
```

Pauses the program for *n* milliseconds.

3.6.11 RefreshMap

```
RefreshMap()
```

Redraws the map. After making changes to objects that will result in changes to the display of the map (adding objects, moving objects, etc), you can call RefreshMap() to update the display. Note that some events (such as OnZoom) will refresh the map anyways, so you will not always have to use this function to update the display.

3.6.12 ScreenToWorldX

```
ScreenToWorldX( x: integer): integer
```

Takes a horizontal (x) point on the screen, and converts it to it's world value. For example,

ScreenToWorldX(200) returns a value that tells you what world x value (on the 0-10,000,000 map scale) that point represents.

3.6.13 ScreenToWorldY

```
ScreenToWorldY( y: integer): integer
```

Takes a vertical (y) point on the screen, and converts it to it's world value. For example, ScreenToWorldY(200) returns a value that tells you what world y value (on the 0-10,000,000 map scale) that point represents.

3.6.14 SetDisplayQuality

```
SetDisplayQuality( nMode: integer)
```

Sets the current display quality - draft, fast, good, highest. For nMode, use one of the following values:

```
0 - Draft
1 - Fast
2 - Good
3 - High
```

3.6.15 SetMapZoom

```
SetMapZoom( n: float)
```

Sets the current map zoom setting to n%.

3.6.16 SetTimer

```
SetTimer( n: integer)
```

Enables the program's internal timer. When n is greater than zero, the timer is enabled, and OnTimer events will be fired every n milliseconds. To set the interval to 5 seconds, for example, pass in 5000 as the parameter. To enable the timer as soon as the map is loaded, set the timer in the OnLoad event.

3.6.17 StatusMsg

```
StatusMsg( s: string)
```

Sets the message displayed in the program's status bar to *s*.

3.6.18 SuppressFileLoadWarnings

```
SuppressFileLoadWarnings( b: boolean)
```

Toggles the current suppression of warning dialog boxes during file loading. When a map with a version problem is loaded - for example, opening a map made in a version of the software that's later than the one you're using - a dialog is displayed warning that the file may be in a format the program can't read. By toggling the warning suppression, you can prevent these warning dialogs from being displayed when files are loaded by your scripts. Pass true to suppress the warnings, false to enable them.

3.6.19 UpdateLayerDisplay

```
UpdateLayerDisplay()
```

Updates the list of layer names in the layer drop down on the program's toolbar after layer names have been changed in a script.

3.6.20 WorldToScreenX

```
WorldToScreenX( x: integer)
```

Takes a horizontal (x) 'world' value, and returns where (in pixels) that value would be on the map display. Note that WorldToScreenX() will often return values that are out of the range of the screen (negative coordinates) due to scrolling, zooming, etc. Use GetScreenHeight() and GetScreenWidth() to determine if a pixel position is visible.

3.6.21 WorldToScreenY

```
WorldToScreenY( y: integer)
```

Takes a vertical (y) 'world' value, and returns where (in pixels) that value would be on the map display. Note that WorldToScreenY() will often return values that are out of the range of the screen (negative coordinates) due to scrolling, zooming, etc. Use GetScreenHeight() and GetScreenWidth() to determine if a pixel position is visible.

3.6.22 ZoomToFit

```
ZoomToFit()
```

Sets the zoom level of the currently displayed map so that the entire map fits within the display window.

3.7 MapObject Class

The MapObject class represents the objects that get placed onto a map. The program maintains an internal array of the MapObjects that are on your map. By using the GoblinAPI, you can access the objects stored within the internal array, and manipulate the them to do your evil bidding!

3.7.1 Related Functions

3.7.1.1 CreateMapObject

```
CreateMapObject( sType: string): MapObject
```

Creates and returns a MapObject of type *sType*. The valid MapObject types are:

- "Polygon" (This is the primitive used for all polygons in their fractal and non-fractal forms. This includes rectangles, land masses, freehand land masses, etc.)
- "Polyline" (This is the primitive used for all line based objects other than splines. This includes fractal lines, freehand lines, etc.)
- "Polypath" (a path object)
- "Spline" (a non-closed spline curve)
- "ClosedSpline"
- "Circle", "Ellipse"
- "Hallway"
- "SmartBuilding"
- "Submap" (Use for creating vectored symbols)
- "Symbol" (Use for creating raster symbols - .PNG, bitmap, etc)
- "Text"

Errors in object creation (if a parameter is incorrect, for example) are accessible with the MapperErrorCode() function. You can alternatively test for the successful creation of a MapObject using the NotNull() function.

Example 1:

```
Dim o, oMap
oMap = GetCurrentMap()
o = CreateMapObject( "Polygon")
If MapperErrorCode = 0 Then
   o.AddPoint 100, 300
   o.AddPoint 100, 400
   o.AddPoint 200, 400
   '...add more points...
   oMap.AddObject o
End If
```

Example 2 (using NotNull()):

```
Dim o, oMap
oMap = GetCurrentMap()
o = CreateMapObject( "Polygon")
If NotNull( o) Then
```

```
      o.AddPoint 100, 300
      o.AddPoint 100, 400
      o.AddPoint 200, 400
      '...add more points...
      oMap.AddObject o
   End If
```

3.7.1.2 GetLastMapObject

```
GetLastMapObject(): MapObject
```

Returns the MapObject object that represents the last object placed on the map, or null if no objects are on the map.

Example:

```
   Dim o
   o = GetLastMapObject()
   If NotNull( o) Then
      ... Do things...
   End If
```

3.7.1.3 SetMapObjectSpecs

```
SetMapObjectSpecs( o: MapObject)
```

This function sets the passed MapObject's properties to match the settings currently selected in the user interface. For example, if you create a rectangle in your script and then call this function, the rectangle's colors will be set to whatever the currently selected colors in the user interface are, the line style will be set according to the current line style selections, etc.

3.7.1.4 SetMaxNodes

```
SetMaxNodes( n: integer): integer
```

Overrides the internal limit on the maximum number of nodes added to an object during the fractalization routines, setting the value to *n*. Return value is the limit that was set prior to calling this function. Store the return value so that you can restore the program back to using the default setting.

The initial inclination might be to set this number to a very high number of nodes. But, this function can be equally useful in creating objects that have very low numbers of nodes created by the fractal routines.

3.7.2 Properties

3.7.2.1 Alpha

```
Alpha: integer
```

This specifies the level of transparency to use when drawing the object. This value can range from 0 (transparent) to 255 (default - solid).

3.7.2.2 ArrowEndStyle

```
ArrowEndStyle: integer
```

Specifies the arrow style at the end of a line. The valid values are:

```
0 - No Arrow
1 - Standard Arrow Head
2 - Flat Head
3 - Arrow w/ flat head;
4 - Circle about end point
5 - Square centered on end point
6 - Reverse arrow
7 - Diamond, centered
8 - Arrow end
9 - Double arrow end
10 - Solid arrow end
```

3.7.2.3 ArrowSize

```
ArrowSize: integer
```

This is the size of an object's arrow tip, if any.

3.7.2.4 ArrowStartStyle

```
ArrowStartStyle: integer
```

Specifies the arrow style at the start of a line. The valid values are:

```
0 - No Arrow
1 - Standard Arrow Head
2 - Flat Head
3 - Arrow w/ flat head;
4 - Circle about end point
5 - Square centered on end point
6 - Reverse arrow
7 - Diamond, centered
8 - Arrow end
9 - Double arrow end
10 - Solid arrow end
```

3.7.2.5 BaseType

```
BaseType: string
```

This is the base type of the map object. Valid types are:

```
"Polygon" (This is the primitive used for all polygons in their fractal and non-fra
"Polyline" (This is the primitive used for all line based objects other than spline
```

```
"PolyPath"
"Spline"
"Closed Spline"
"Arc"
"Circle"
"Hallway"
"Submap" (This is a vectored symbol)
"Symbol" (This is a bitmap or jpeg symbol)
"Text"
"Smart Building" or "PolyBuilding"
```

3.7.2.6 BevelLightAngle

```
BevelLightAngle: integer
```

This is the angle from which the object's bevel is lit.

3.7.2.7 BevelLightIntensity

```
BevelLightIntensity: integer
```

This is the lighting intensity of the object's beveled edge.

3.7.2.8 BevelStyle

```
BevelStyle: integer
```

The bevel style of an object, if any. Valid values are:

```
0 - None
1 - Flat
2 - Rounded
3 - Sloped Up
4 - Stepped
5 - Small Steps
6 - Frame
```

3.7.2.9 BevelWidth

```
BevelWidth: integer
```

This is the width of the object's beveled edge.

3.7.2.10 BitmapFile

```
BitmapFile: string
```

This is the name of the image file assigned to the object. This is only used by raster mapping symbol objects.

3.7.2.11 BlockText

```
BlockText: string
```

This is the object's Block Text notes. This is used by the Scenario Builder when compiling a document.

3.7.2.12 BlurStyle

```
BlurStyle: integer
```

This is the object's blur style. Valid values are:

```
0 - None (removes the blur)
1 - Standard (fast bi-directional box blur)
2 - Feathered Edge
3 - Inverse Feather
```

3.7.2.13 BlurWidth

```
BlurWidth: integer
```

This is the width of the object's blur (if any)

3.7.2.14 BrushColor

```
BrushColor: integer
```

This is the fill color of the object, as represented by a 32bit integer.

3.7.2.15 CanEmbed

```
CanEmbed: boolean
```

Applies to mapping symbol objects. Specifies whether or not the mapping symbol image file or .fmp file can be embedded within the map's file when it is saved.

3.7.2.16 ClearFill

```
ClearFill: boolean
```

Specifies if the object is filled when drawn. If true, the object is clear. If false, the object is drawn with the assigned fill color or pattern.

3.7.2.17 EllipseSegmentRange

```
EllipseSegmentRange: integer
```

This is the range, 0-360, if this object is an ellipse segment.

3.7.2.18 EllipseSegmentStyle

```
EllipseSegmentStyle: integer
```

This is the style of ellipse, if the object is an ellipse. Valid values are:

```
0 - Chord
1 - Pie Segment
```

3.7.2.19 FillPattern

```
FillPattern: string
```

This names the fill pattern to use when drawing the object. The value is the name of the fill pattern, such as 'Red Tiles 1' or 'Water'. The fill pattern name is the same as the name of the pattern bitmap file that's located in the "\FillPatterns" directory, sans path and file extension.

3.7.2.20 FillPatternBrightness

```
FillPatternBrightness: integer
```

This is the brightness of the fill pattern. values range from 1 to 200, with 100 being the image as-is.

3.7.2.21 FillPatternScale

```
FillPatternScale: integer
```

The scale at which the fill patterns are drawn. Typical values range 1-300, with 100 being 100%.

3.7.2.22 FillStyle

```
FillStyle: integer
```

The style used to fill the object, if a fill pattern is not applied. This is either solid, clear, or a hatch value:

```
0 - Solid
1 - Clear
2 - Horizontal Hatch
3 - Vertical Hatch
4 - Forward Diagonal
5 - Backwards Diagonal
6 - Cross Hatch
7 - Diagonal Cross Hatch
```

3.7.2.23 Flipped

```
Flipped: boolean
```

Controls whether or not the mapping object is drawn flipped - with all it's points flipped about the x axis. True means it is flipped, false (default) means it's not.

3.7.2.24 FontBold

```
FontBold: boolean
```

Specifies if Text objects should be drawn in a bold font. True means yes, false means no.

3.7.2.25 FontColor

```
FontColor: integer
```

This is the color of the font used to draw text objects. The value of this property is a standard 32bit color value.

3.7.2.26 FontItalic

```
FontItalic: boolean
```

Controls whether or not text objects are drawn in an italic font. True is yes, false is no.

3.7.2.27 FontName

```
FontName: string
```

This is the name of the actual face name to use when drawing a text object. E.g., "Times New Roman", "Arial", "Copperplate Gothic", etc.

3.7.2.28 FontSize

```
FontSize: integer
```

This is the size, in world units, of the font to use when drawing text objects.

3.7.2.29 FontUnderline

```
FontUnderline: boolean
```

Controls whether or not text objects are drawn underlined. True is yes, false is no.

3.7.2.30 FractalFactor

```
FractalFactor: integer
```

This is the level of 'fractalization' that should be applied to the object when rendering it. This value ranges from 0 to 10. Rectangles, triangles, and straight lines have a FractalFactor of 0. Very jagged polygons and lines have FractalFactor's of 9 or 10.

3.7.2.31 GradientAlpha

```
GradientAlpha: integer
```

The transparency value of the gradient color. Ranges from 0 (transparent) to 255 (solid).

3.7.2.32 GradientColor

```
GradientColor: integer
```

This is the 2nd color used when applying a gradient to an object.

3.7.2.33 GradientStyle

```
GradientStyle: integer
```

Specifies the style of gradient to fill the object with, if any:

```
0 - None
1 - Linear
2 - Radial
```

3.7.2.34 GridAlpha

```
GridAlpha: integer
```

This is the alpha value of the map's grid color, ranging from 0 (transparent) to 255 (completely solid)

3.7.2.35 GridColor

```
GridColor: integer
```

Specifies the color of the map's grids (if they are displayed).

3.7.2.36 GridFont

```
GridFont: string
```

This is the font face name used when labeling / numbering grids within the object. Example: 'Arial', 'Copperplate Gothic Bold', etc.

3.7.2.37 GridFontSize

```
GridFontSize: integer
```

This is the font size, in the mapper's world units, to use when labeling grids that fill the object.

3.7.2.38 GridHeight

```
GridHeight: integer
```

This is the height of the object's grids, in world units.

3.7.2.39 GridLabelAlign

```
GridLabelAlign: integer
```

This specifies the label alignment when grids used as a fill are numbered. The following values are valid:

```
0 - Top
1 - Center
2 - Bottom
```

3.7.2.40 GridNumbering

```
GridNumbering: boolean
```

If set to true, then grid numbering/labeling for grids used as a fill within the object is enabled. If false, no grid numbers are displayed.

3.7.2.41 GridSync

```
GridSync: boolean
```

Specifies if the object's grid fill is synchronized with the main map's grid.

3.7.2.42 GridType

```
GridType: integer
```

Specifies the type of grid to fill the object with, if any. The possible values are:

```
0 - No grids (default)
1 - Square
2 - Hex
3 - Dot
4 - Offset Square
```

3.7.2.43 GridWidth

```
GridWidth: integer
```

This is the width of the map's grids, in world units.

3.7.2.44 GroupID

```
GroupID: integer
```

This is the object's GroupID. If the object is grouped with other objects, they will share GroupID's.

3.7.2.45 Layer

```
Layer: integer
```

Indicates which layer is assigned to the object. This will be an integer between 0 - 255.

3.7.2.46 LineScaled

```
LineScaled: boolean
```

Specifies whether or not the width of the line should be scaled according to the zoom factor of the map. If true, the line widens as you zoom in on the map. If false, the line with always remains at the value assigned to LineThickness.

3.7.2.47 LineThickness

```
LineThickness: integer
```

This is the width in pixels of the line.

If the object's LineScaled property is true, this is the width, in world units, of the line.

3.7.2.48 Locked

```
Locked: boolean
```

Controls whether or not the object can be manually moved by the user (within the standard user interface). If true, the user cannot move the object. If false, the user can delete or move the object. This does not apply to deleting or moving objects from within scripts.

3.7.2.49 MapLink

```
MapLink: string
```

This is the target of the object's link, if any. MapLink can be the name of another Fractal Mapper file, the name of a saved view, the name of an external application file (such as a PDF), or a URL. MapLinkStyle determines how this attribute is used.

3.7.2.50 MapLinkStyle

```
MapLinkStyle: integer
```

Specifies whether the MapLink property points to a file or a saved view.

Possible values are:

```
0 - File or URL
1 - Saved View
```

3.7.2.51 Mirrored

```
Mirrored: Boolean
```

Specifies if the object is to be drawn mirrored - with all it's points swapped along the y axis. If true, it is mirrored. If false, it is not.

3.7.2.52 NoteName

```
NoteName: string
```

This is the title associated with the object's notes.

3.7.2.53 NotePictureOutputWidth

```
NotePictureOutputWidth: integer
```

If NotePictureStyle is anything other than 0 (meaning an illustration is exported for this object in Scenario Builder documents), this values sets the width, in pixels, of the illustration image.

3.7.2.54 NotePictureStyle

```
NotePictureStyle: integer
```

This is the style of illustration to output when the Scenario Builder is run. The currently supported styles are:

```
0 - None (no illustration is output)
1 - Selected Area (the area selected
    by the user on the notes window
    is exported as an illustration)
```

3.7.2.55 Notes

```
Notes: string
```

This is the notes assigned to the object.

3.7.2.56 ObjectName

```
ObjectName: string
```

This is an internal name that can be used for the object.

3.7.2.57 PenColor

```
PenColor: integer
```

This is the color that is used when drawing lines (or the outlines on filled objects).

3.7.2.58 RandomSeed

```
RandomSeed: integer
```

An integer that is used as a seed for the internal pseudo-random number generator when rendering fractal objects (objects with FractalFactors greater than 0). Changing this value will have the same effects as applying the 're-render' menu option to an object on the map. This is ignored when drawing objects with a 0 FractalFactor.

3.7.2.59 Rendered

```
Rendered: boolean
```

This property reports whether the object has been 'rendered' by the mapper's graphics engine. Do not change the value of this property in your scripts. To set the Rendered property to false, call the Unrender() method. Setting the value of this to true on unrendered objects can lead to access violations.

3.7.2.60 Rotation

```
Rotation: integer
```

Specifies the rotation, in degrees, that the object should be drawn at. The valid range is 0 (no rotation) to 359.

3.7.2.61 Selected

```
Selected: boolean
```

This reports or sets the object's 'selected' state (for example, if it is clicked on with the pointer tool in the user interface and is currently selected).

Example:

```
'Changes the brush color of all the
'currently selected objects
'on the map to red
oMap = GetCurrentMap()
For i = 1 To oMap.ObjectCount()
   o = oMap.GetObject( i)
   If o.Selected Then
      o.BrushColor = rgb( 255,0,0)
      o.Unrender
   End If
Next
```

3.7.2.62 ShadowAngle

```
ShadowAngle: integer
```

This the angle of the light source of the object's special effects shadow, if any. Range is 0-360.

3.7.2.63 ShadowColor

```
ShadowColor: integer
```

This is the color of the object's special effects shadow.

3.7.2.64 ShadowOffset

```
ShadowOffset: integer
```

This is the offset, in pixels, of the object's special effect's shadow. Typical values are 1-100.

3.7.2.65 ShadowStyle

```
ShadowStyle: integer
```

This is the type of special effects shadow applied to the object, if any.

Valid values are:

```
0 - No Shadow (removes shadow)
1 - Drop Shadow
2 - Inner Shadow
```

3.7.2.66 ShadowWidth

```
ShadowWidth: integer
```

This is the width of the blur of the object's special effects shadow, if any. Typical values range from 1-100.

3.7.2.67 SubmapFile

```
SubmapFile: string
```

If this object is a Submap Object (i.e., a vectored symbol), this property specifies the name of the vectored symbol to use.

Example:

```
' Changes the Submap Object's symbol to Tree1.fmp
o = GetLastMapObject()
o.SubmapFile = "MapArt\Standard Symbols\Trees\Tree1.fmp"
```

3.7.2.68 SubmapIndex

```
SubmapIndex: integer
```

To increase efficiency, submap objects (vectored symbols) are loaded only once, regardless of how many times on a map (the number of instances) the submap is displayed. When a submap object is loaded, it is placed into a pool of submap resources in memory. These submap resources are accessible as an array.

Each instance of a vectored symbol on a map will have an associated SubmapIndex value. This is the position in the internal resources array where this particular submap was loaded. Each time an instance of that submap is drawn on the map, it's data (the polygons, lines, etc that make up the symbol) is pulled from the resource array.

For example, say you have 50 tree symbols on your map. This is 50 MapObject objects. Each MapObject object will have the same SubmapIndex value. This SubmapIndex value will contain the

index position of that specific .fmp (for example, 'tree3.fmp') within the resources array.

The ability to access the submap resource array means you can access all the MapObjects stored within a symbol, make changes, and even save them back to their files.

The SubmapIndex property is used with the GetSubmapFromResources() global function.

Example:

```
'places a vectored symbol at position x,y, and sets the default layer.
o = map.PlaceSubmapAtScale( "MapArt\Trees\Tree.fmp", x, y)
If NotNull( o) Then
   'get a Map object that represents the polygons in the tree.
   os = GetSubmapFromResources( o.SubmapIndex)

   'set the layer, default to 0
   l = 0
   If NotNull( os) Then
      l = os.SubmapDefaultLayer
   End If
   o.Layer = l
End If
```

3.7.2.69 Tag

```
Tag: integer
```

The Tag property is a temporary holding place for an integer that you wish to associate with the object. For example, if you are toggling the BrushColor on each timer event for animation effects, you could store the toggled color in this property. In most cases, you'll want to use the custom field system for arbitrary storage of data you wish to be accessible between file loads.

3.7.2.70 Text

```
Text: string
```

This is the text that is displayed when drawing a text object. See the font properties for information on how this text is drawn.

3.7.2.71 TextAlign

```
TextAlign: integer
```

This defines the text alignment that is used when displaying text objects with multiple lines. The valid values for this property are:

```
0 - Left Alignment (default)
1 - Center Alignment
2 - Right Alignment
```

3.7.2.72 TextureFile

```
TextureFile: string
```

This is the name of the grayscale image file being used as a texture source by the object.

3.7.2.73 TextureIntensity

```
TextureIntensity: integer
```

This is the intensity at which the object's texture, if assigned, is applied. Normal values range from 0 to 100.

3.7.2.74 TextureScale

```
TextureScale: integer
```

This is the scale at which the object's texture is applied. Typical values range between 1 and 100.

3.7.2.75 Transparent

```
Transparent: boolean
```

This value controls whether or not a raster based mapping symbol object is drawn with a transparent background or not. In most cases, this value is true (object is drawn with a transparent background). In certain cases, such as border graphics, the transparent effect may not be desired, and this value is set to false. Note that this is NOT the same as the object's Alpha setting, and does not affect any alpha channels embedded within a .PNG image. This applies only to images without an alpha channel.

3.7.2.76 ZoomCeiling

```
ZoomCeiling: integer
```

If set to a value greater than zero, this specifies that the object will not be drawn when the zoom factor is greater than this value. For example, if set to 45, the object will not draw when the zoom factor is greater than 45%. This is useful in eliminating large objects (like country names) when zooming in.

3.7.2.77 ZoomFloor

```
ZoomFloor: integer
```

If set to a value greater than zero, this specifies what the minimum zoom factor is for displaying the object on the map. For example, if this value is set to 10, then the map object will not be displayed unless the current zoom factor is at least 10%. This is useful, for example, when hiding 'detail' items such as town and village names when zoomed out (leaving just the country and region names displayed).

3.7.3 Methods

3.7.3.1 AddPaletteReplacement

```
AddPaletteReplacement( c: integer, r: integer)
```

This method adds a Palette replacement (substitution) to a Submap object (a vectored symbol). When a palette replacement is added, any instance of color *c* in the object is changed to *r*. This is similar in effect to the palette replacement bar at the bottom of Fractal Mapper.

3.7.3.2 AddPoint

```
AddPoint( x: integer, y: integer)
```

Adds the point (x,y) to the map object. To create a rectangle, for example, you'd add four points.

Example:

```
'Creates a rectangle and adds it to the map.
Dim o
o = CreateMapObject( "Polygon")
o.FractalFactor = 0
o.AddPoint 4000, 4000
o.AddPoint 4000, 6000
o.AddPoint 6000, 6000
o.AddPoint 6000, 4000
AddMapObject o
```

3.7.3.3 CheckRasterRendered

```
CheckRasterRendered()
```

Renders an object that has special effects applied to it (blur, bevel, etc), or clears any memory previously used for special effects rendering if the effects have been removed.

3.7.3.4 ClearPaletteReplacements

```
ClearPaletteReplacements()
```

This method clears any Palette Replacements that have been assigned to a Submap Object (vectored symbol) by either the AddPaletteReplacement() method, or through the program's user interface. When palette replacements are removed, the symbol will display with the colors defined in it's file.

3.7.3.5 ClearRasterEffects

```
ClearRasterEffects()
```

Removes all special effects (bevels, blurs, etc) applied to an object.

3.7.3.6 ClickedOn

```
ClickedOn( x: integer, y: integer): boolean
```

Returns whether or not the point (x,y) resides within the object.

3.7.3.7 CloneTo

```
CopyTo( o: MapObject)
```

Copies the object's properties (properties, points, notes, custom fields, and all) to another Map object. The MapObject being copied to must be a valid Map object (created with the CreateMapObject() function or one that already exists on the map).

3.7.3.8 CloneToMask

```
CloneToMask( var dest: MapObject)
```

Returns a copy of the object with all the colors set to black and with all the effects removed.

3.7.3.9 CopyTo

```
CopyTo( o: MapObject)
```

Copies the object's properties (properties, points, notes, custom fields, and all) to another MapObject. The MapObject being copied to must be a valid MapObject (already created with the CreateMapObject () function or one that already exists on the map).

3.7.3.10 DeleteAllNodes

```
DeleteAllNodes()
```

Removes all points from an object.

3.7.3.11 DeleteNode

```
DeleteNode( n: integer)
```

Delete's point *n* from the object. Do not call this against a complex polygon (a polygon with more than one part).

3.7.3.12 GetField

```
GetField( s: string): string
```

Retrieves the value of the object level custom field named *s*.

Example:

```
'iterates through all the objects, looking for the
'one with an object custom field called
''Last Modified', and sets the text of that
'object to the value of the 'Last Modified On'
'map level custom field.
```

```
Sub OnLoad()
   Dim s, n, i, o, f
   s = GetMapField( "Last Modified On")
   n = GetMapObjectCount()
   For i = 1 To n
      o = GetMapObject( i)
      f = o.GetField( "Last Modified")
      If Len( f) > 0 Then
         f.Text := s
      End If
   Next
End Sub
```

3.7.3.13 GetFieldByIndex

```
GetFieldByIndex( n: integer): string
```

Returns the value of the object's *n*-th custom field, or an empty string if there is no *n*-th custom field.

3.7.3.14 GetFieldCount

```
GetFieldCount(): integer
```

Returns the number of custom fields that have been assigned to the object.

3.7.3.15 GetFieldNameByIndex

```
GetFieldNameByIndex( n: integer): string
```

Returns the name of the object's n-th custom field, or an empty string if there is no field at that index.

3.7.3.16 GetGradientControlPointX

```
function GetGradientControlPointX( nIndex: integer): integer
```

Returns the X portion of a gradient control point. *nIndex* specifies which gradient control point you're referencing. This can be 0 or 1. 0 is the first point (where the color is set to BrushColor). 1 is the second point (where the color is set to GradientColor).

3.7.3.17 GetGradientControlPointY

```
function GetGradientControlPointY( nIndex: integer): integer
```

Returns the Y portion of a gradient control point. *nIndex* specifies which gradient control point you're referencing. This can be 0 or 1. 0 is the first point (where the color is set to BrushColor). 1 is the second point (where the color is set to GradientColor).

3.7.3.18 GetPaletteReplaceColor

```
GetPaletteReplaceColor( c: integer): integer
```

If a color replacement has been assigned to color *c*, this returns the replacement color. Otherwise returns *c*.

3.7.3.19 GetPointCount

```
GetPointCount(): integer
```

This returns the number of draw points stored for the object (the points the user added before the object was fractalized). You can use this in conjunction with GetPointX() and GetPointY() to navigate through an object's points.

3.7.3.20 GetPointX

```
GetPointX( n: integer): integer
```

This returns the 'X' value (horizontal) of the object's *n*-th draw point.

Example:

```
'Centers the map on the object's first point
o = GetMapObject( 1)
...
x = o.GetPointX( 1)
y = o.GetPointY( 1)
SetMapZoom 45
CenterMapOnXY x, y
```

3.7.3.21 GetPointY

```
GetPointY( n: integer): integer
```

This returns the 'Y' value (vertical) of the object's *n*-th draw point.

Example: (see the GetPointX() method's example)

3.7.3.22 GetRenderPointCount

```
GetRenderPointCount(): integer
```

This returns the number of rendered points stored for the object. You can use this in conjunction with GetRenderPointX() and GetRenderPointY() to navigate through an object's rendered points.

Example:

```
'creates a duplicate of a polygon, only twice the size,
'by reading a polygon's render points, and creating a
'new polygon
```

```
Sub TwiceAsBig
   Dim i, n, x, y
   Dim o, o2
   On Error Resume Next
   o = GetMapObject(1)   'just use 1 here for demo purposes
   n = o.GetRenderPointCount()
   o2 = CreateMapObject( "Polygon")
   For i = 1 To n
      x = o.GetRenderPointX(i)
      y = o.GetRenderPointY(i)
      x = x * 2
      y = y * 2
      o2.AddPoint x, y+2000
   Next
   o2.BrushColor = o.BrushColor
   o2.PenColor = o.PenColor
   AddMapObject o2
   RefreshMap
End Sub
```

3.7.3.23 GetRenderPointX

```
GetRenderPointX( n: integer): integer
```

This returns the 'X' value (horizontal) of the object's *n*-th point.

Example: (see the GetRenderPointCount() method's example)

3.7.3.24 GetRenderPointY

```
GetRenderPointY( n: integer): integer
```

This returns the 'Y' value (vertical) of the object's *n*-th point.

Example: (see the GetRenderPointCount() method's example)

3.7.3.25 HasRasterEffects

```
HasRasterEffects: boolean
```

Returns true if the object has any raster (special) effects (bevels, textures, blurs, shadows) applied.

3.7.3.26 isInZoomWindow

```
isInZoomWindow( z: integer): boolean
```

Determines if the value z is within the 'zoom window' created by the ZoomFloor and ZoomCeiling properties. Returns true if the it is, false if it is not (meaning the object shouldn't be drawn).

3.7.3.27 Move

```
Move( dx:integer, dy: integer)
```

Moves the object by dx units horizontally and dy units vertically. Positive values move to the right and bottom, negative values move the object to the left and top.

3.7.3.28 Overlaps

```
Overlaps( o: MapObject): boolean
```

Returns true if the map object (the one whose method your are calling) and the map object *o* (the one passed as a parameter) overlap, false if they do not.

3.7.3.29 PaletteReplacementIndex

```
PaletteReplacementIndex(c: integer): integer
```

Returns the index at which color c resides within an object's Palette Replacements (see AddPaletteReplacment()) array, or -1 if it is not found. Useful for deciding whether a particular color has already been assigned to a Submap Object's color replacement list.

3.7.3.30 RasterRender

```
RasterRender()
```

Renders any applied special effects. Normally this would occur automatically by the drawing process, but its possible to render via script for cases where a map is being generated in-memory (i.e., a map other than the main map).

3.7.3.31 RasterUnrender

```
RasterUnrender()
```

Marks any special effects applied to the object as un-rendered.

3.7.3.32 Render

```
Render()
```

Processes the object in preparation for drawing. The rendering process performs any necessary fractalization, loading resource graphics, etc. This normally is performed automatically by the drawing engine, but can be called in code manually when working on a map in-memory. This only should be called if the object's Rendered property is false.

3.7.3.33 Sample

```
Sample( n: integer)
```

If the object has more than *n* nodes, this method reduces the number of nodes in the object to approximately *n*. This works off of the rendered/fractally generated Render points. If the object is re-rendered, it's node count returns to the original number. Typically you'd want to do something with the nodes right after an object was sampled - before they revert back (for example, copy it to another object).

3.7.3.34 ScaleObject

```
ScaleObject( rX, rY: float; doPropScale: boolean)
```

Scales the object about it's center mass by a factor of *rX* horizontally, and *rY* vertically. If *doPropScale* is true, the scale is performed proportionally based on the *rX* factor.

3.7.3.35 ScalePrecision

```
ScalePrecision( OldSize: integer, NewSize: integer)
```

Scales all the points within an object by a factor of *NewSize*/*OldSize*.

3.7.3.36 ScaleXY

```
ScaleXY( sx, sy: float)
```

Scales the object by a factor of *sx* horizontally, and *sy* vertically, maintaining its top-left position. Use this when scaling more than one object at a time to maintain their relative positions to each other.

3.7.3.37 SetField

```
SetField( s: string, v: string)
```

Creates custom field within the object, called *s*, and sets it's value to *v*. If a custom field by the same name already exists, it's value is replaced by *v*. To delete a field, set the value to an empty string. Note that the object's custom field functions differ from the map's custom field functions. Fields set with the object level methods (this one) are assigned to individual objects, whereas fields set with SetMapField () are assigned to the Map as a whole.

3.7.3.38 SetGradient

```
SetGradient( c: integer, mode: integer)
```

Sets the object's gradients. In additon - perhaps more importantly - this also sets the gradient control points on the object if none have been set already. To set the gradient Alpha value, use the

GradientAlpha property.

3.7.3.39 SetGradientControlPoints

```
SetGradientControlPoints( x1: integer, y1: integer, x2: integer, y2: integer)
```

Sets the control points to use if the object is filled with a gradient. Point (x1, y1) determines the start of the gradient, and is filled with the BrushColor, and point (x2, y2) is the end point of the gradient, and is set to GradientColor. All other points within the object are set to a color blend based on the position of the control points and the type of the gradient in use.

3.7.3.40 SetNotePictureRect

```
SetNotePictureRect( l, t, r, b: integer)
```

Assigns the region of the map to be used as an illustration for this object when the scenario builder is run. The points *l*, *t*, *r*, and *b* represent the left, top, right, and bottom coordinate, in world units.

3.7.3.41 SetPoint

```
SetPoint( n: integer, x: integer, y: integer)
```

Sets coordinates of the object's *n*-th draw point to (x,y). Use this to change the shape of an object. To actually move an object, use the Move() method.

3.7.3.42 SetRotate

```
SetRotate( a: integer)
```

Sets the object's rotation setting to the angle *a.*, If *a* is below 0 or greater than 359, this method adjusts the value to the corresponding angle (i.e., it performs a modulus operation).

3.7.3.43 SimplifyNodes

```
SimplifyNodes()
```

Removes any consecutive duplicate nodes from an object. Call the object's Unrender() method after using this method.

3.7.3.44 Subdivide

```
Subdivide()
```

Inserts additional nodes between existing nodes. This adds many thousands of points (approx 8000 by default in FM), creating an object with many nodes (much like a cloned fractal object). Call Unrender() after using this method.

3.7.3.45 UnRender

```
UnRender()
```

Sets the Rendered attribute of the object to false. Call this function after changing it's node positions or when changing any properties which affect the object's shape (including rotation & mirroring), size, or patterns.

3.8 Map Class

The Map Class represents both entire maps and individual submaps (vectored symbols). You can use objects of type Map to load maps from file, add and delete objects on them, access the map settings (grids, background colors, etc), and save them back to a file.

3.8.1 Related Functions

3.8.1.1 GetCurrentMap

```
GetCurrentMap(): Map
```

Returns a map object that represents the currently displayed map.

Example:

```
'Sets the current map's background color to white
Sub Main
   oMap = GetCurrentMap()
   oMap.BackgroundColor  = RGB( 255,255,255)
End Sub
```

3.8.1.2 NewMap

```
NewMap(): Map
```

Creates and returns a new map object.

Example (see FreeMap())

3.8.1.3 FreeMap

```
FreeMap( o:Map)
```

Deletes a Map from memory and frees resources. Do **not** call this on the currently displayed map - only use this with maps created in code.

Example:

```
'Makes a new map, adds a random landmass to it,
'saves it to a file, and finally clears it from memory
```

```
Sub Main
   'Make a new map
   oMap = NewMap()
   oMap.BackgroundColor = RGB( 193,224,255)

   'Make a new land mass
   oLM = CreateMapObject( "Polygon")
   oLM.FractalFactor = 6
   oLM.BrushColor = RGB( 200, 215, 200)
   oLM.AddPoint 2000000, 2000000
   oLM.AddPoint 2000000, 8000000
   oLM.AddPoint 8000000, 8000000
   oLM.AddPoint 8000000, 2000000
   oMap.AddObject oLM
   oMap.SaveToFile "c:\temp\testmap.fmp"
   FreeMap oMap
End Sub
```

3.8.1.4 GetSubmapFromResources

```
GetSubmapFromResources( nIndex: integer): Map
```

To increase efficiency, submap objects (vectored symbols) are loaded only once, regardless of how many times on a map (the number of instances) the submap is displayed. When a submap object is loaded, it is placed into a pool of submap resources in memory. These submap resources are accessible as an array.

Each instance of a vectored symbol on a map will have an associated SubmapIndex value. This is the position in the internal resources array where this particular submap was loaded. Each time an instance of that submap is drawn on the map, it's data (the polygons, lines, etc that make up the symbol) is pulled from the resource array. This data is retrieved from the resources array by using the GetSubmapFromResources() function.

For example, say you have 50 tree symbols on your map. This is 50 MapObject objects. Each MapObject object will have the same SubmapIndex value. This SubmapIndex value will contain the index position of that specific .fmp/.skt (for example, 'tree3.fmp') within the resources array.

The ability to access the submap resource array means you can access all the MapObjects stored within a symbol, make changes, and even save them back to their files.

Example:

```
'places a vectored symbol at position x,y, and sets the default layer.
o = map.PlaceSubmapAtScale( "MapArt\Trees\Tree.fmp", x, y)
If NotNull( o) Then
   'get a Map object that represents the polygons in the tree.
   os = GetSubmapFromResources( o.SubmapIndex)

   'set the layer, default to 0
   l = 0
   If NotNull( os) Then
      l = os.SubmapDefaultLayer
   End If
   o.Layer = l
End If
```

3.8.2 Properties

3.8.2.1 BackgroundColor

```
BackgroundColor: integer
```

This is the background color of the Map, as expressed by an integer. Use VBScript's RGB() function to create this color from a combination of Red, Green, and Blue values.

3.8.2.2 BlockText

```
BlockText: string
```

This is the block text note (GM's note) assigned to the map.

3.8.2.3 ColorMode

```
ColorMode: integer
```

This defines the current color mode (color theme) the map is displayed in. The possible values are:

```
0 - Full Color
1 - All Black (all colors as black, for mask creation)
2 - Black and White (raster images displayed in grayscale)
3 - Grayscale
4 - Sepia Tone
```

3.8.2.4 FileName

```
FileName: string
```

This is the name of the currently opened file. Will be empty if file is new and has not been saved yet.

3.8.2.5 FormatVersion

```
FormatVersion: integer
```

This is the file format version the map was stored in. This may or may not be the same as the version of the program.

3.8.2.6 GridAlpha

```
GridAlpha: integer
```

This is the Alpha (transparency) level component in the map's grid color. Range is 0-255.

3.8.2.7 GridColor

```
GridColor: integer
```

This is the color of the map's grids, in integer form.

3.8.2.8 GridFont

```
GridFont: string
```

This is the name of the font to use when labeling grid cell numbers.

3.8.2.9 GridFontSize

```
GridFontSize: integer
```

This is the map's grid font size, in world units.

3.8.2.10 GridHeight

```
GridHeight: integer
```

This is the height of the map's grids, in world units. Use Map.UserToWorld() and Map.WorldToUser() to convert back and forth between user units (feet, meters, etc) and world units.

3.8.2.11 GridLabelAlign

```
GridLabelAlign: integer
```

This specifies the text alignment of grid cell labels. The possible values are:

```
0 - Top
1 - Center
2 - Bottom
```

3.8.2.12 GridLocation

```
GridLocation: integer
```

If grids are displayed on the map, this property sets the location of the grids in relation to the objects on the map. Possible values are:

```
   0 - Grids on Top (default)
   1 - Grids on Bottom
```

3.8.2.13 GridNumbering

```
GridNumbering: boolean
```

If set to true, then grid numbering/labeling on your map is enabled. If false, no grid numbers are displayed.

3.8.2.14 GridType

```
GridType: integer
```

Specifies the type of grid to draw on the map, if any. The possible values are:

```
0 - No grids (default)
1 - Square
2 - Hex
3 - Dot
4 - Offset Square
```

3.8.2.15 GridWidth

```
GridWidth: integer
```

This is the width of the map's grids, in world units. Use Map.UserToWorld() and Map.WorldToUser() to convert back and forth between user units (feet, meters, etc) and world units.

3.8.2.16 Height

```
Height: integer
```

Specifies the hieght of the map, in the program's internal units. Use Map.UserToWorld() and Map. WorldToUser() to convert back and forth between user units (feet, meters, etc) and world units.

3.8.2.17 isEmbedded

```
isEmbedded: boolean
```

Specifies whether or not this map, if a submap of another map, is embedded into the parent map when saved.

3.8.2.18 IsModified

```
isModified: boolean
```

Is set to *true* if the map has been modified (in the user interface), and *false* if no changes have been made.

3.8.2.19 isSubmap

```
isSubmap: boolean
```

This tells if the map is a submap of another map (i.e., if the map object is a mapping symbol and not

the map itself). Is *true* if the map is a submap.

3.8.2.20 LastSelectedIndex

```
LastSelectedIndex: integer
```

This is the index of the last object on the map to be selected manually (i.e., with the Pointer tool).

3.8.2.21 LastSelectedNode

```
LastSelectedNode: integer
```

This is the index of the last node of the last object to be selected manually (i.e., with the Pointer tool).

3.8.2.22 NotePictureOutputWidth

```
NotePictureOutputWidth: integer
```

Specifies the width, in pixels, of the main map illustration that is created when the scenario builder is run.

3.8.2.23 NotePictureStyle

```
NotePictureStyle: integer
```

Defines whether or not a map illustration is created when the scenario builder is run. Possible values are:

```
0 - No main map illustration
1 - map illustration created
```

3.8.2.24 Notes

```
Notes: string
```

This is the map's notes (player notes).

3.8.2.25 ParentFWEFileName

```
ParentFWEFileName: string
```

This is the background Fractal World Explorer map file for this map, if any.

3.8.2.26 ParentGraphicFileName

```
ParentGraphicFileName: string
```

This is the name of the background graphic (image) file, if any.

3.8.2.27 ParentMapFileName

```
ParentMapFileName: string
```

This is the name of the map's background map file, if any.

3.8.2.28 Precision

```
Precision: integer
```

This value is the size of the Map's the coordinate system in the Fractal Mapper's internal units. For most maps, this will be 10,000,000 (the default size).

3.8.2.29 ResolutionMode

```
ResolutionMode: integer
```

This is the DPI mode for special effects. The possible values are:

```
0 - Low
1 - Middle
2 - High
```

Since changing a map's resolution mode affects needs a corresponding re-rendering of special effects, its best to use the Map's SetResolutionMode() method to set this value.

3.8.2.30 SpanHeight

```
SpanHeight: integer
```

This is the height of the map, in the user's unit of measure (miles, feet, etc.). Use Map.UserToWorld() and Map.WorldToUser() to convert back and forth between user units (feet, meters, etc) and world units.

3.8.2.31 SpanUnitName

```
SpanUnitName: string
```

Specifies the name of the map's unit of measure. Examples: 'miles', 'kilometers', 'meters'. Note that this does not have to be a *real* unit of measure. You can use your own units if desired.

3.8.2.32 SpanWidth

```
SpanWidth: integer
```

This is the map's width, in the user's unit of measure. On a map that is 4000 miles across, this value would be 4000. Use Map.UserToWorld() and Map.WorldToUser() to convert back and forth between user units (feet, meters, etc) and world units.

3.8.2.33 SubmapDefaultLayer

```
SubmapDefaultLayer: integer
```

If the Map object is a Submap (a vectored symbol), the SubmapDefaultLayer value is the index of a Layer that this submap should be placed onto when placed on the map for the first time. When the user adds a submap via the usual user interface, this layer placement occurs automatically. When you place a submap on a map in code, you'd need to perform the layer assignment manually.

Example:

```
o = map.PlaceSubmapAtScale( s, x, y)
If NotNull( o) Then
   os = GetSubmapFromResources( o.SubmapIndex)

   'set the layer, default to 0
   l = 0
   If NotNull( os) Then
      l = os.SubmapDefaultLayer
   End If
   o.Layer = l
End If
```

3.8.2.34 Title

```
Title: string
```

This is the Title of the map. This value is also the title of the map that gets uploaded when you upload the map to NBOS Online Exchange.

3.8.2.35 UploadDescription

```
UploadDescription: string
```

This is the description of the map that gets uploaded when you upload the map to NBOS Online Exchange.

3.8.2.36 Width

```
Width: integer
```

Specifies the width of the map, in the program's internal units. Use Map.UserToWorld() and Map. WorldToUser() to convert back and forth between user units (feet, meters, etc) and world units.

3.8.2.37 Zoom

```
Zoom: float
```

This is the current Zoom level of the map (as displayed on the screen). Note that this value is 1000% smaller than the zoom level that is displayed on the screen. In order to make things conceptually easier for the user, the Fractal Mapper multiplies zoom levels by 1000 before displaying it in the title bar, saved views, etc. When setting a zoom level in code, you must keep this in mind!

3.8.3 Methods

3.8.3.1 AddObject

```
AddObject( o: MapObject): integer
```

Adds object o to the map's list of MapObjects. Returns the index of the MapObject within the map's array of map objects.

3.8.3.2 AddPaletteReplacementSelected

```
AddPaletteReplacementSelected( f, r: integer)
```

Adds a 'Palette Replacement', or color substitution to any selected Submap objects. When a replacement is added, any Submap objects affected will display color f as color r.

3.8.3.3 AddUserPoint

```
AddUserPoint( o: MapObject, X: real, Y: real)
```

Adds a draw point (X, Y) to the MapObject o, in the user's unit of measure. Most other functions work against the Mapper's own internal set of units. This method allows you to add points to an object using the User's unit of measure (meters, miles, etc.). This function actually converts the point (X,Y) to the Mapper's internal units, so once a point is added, it's stored as a point in the Mapper's coordinate system, and will need to be converted back to the user's unit of measurement if accessed via the MapObject Class's GetPointX() and GetPointY() methods.

```
'Creates a dungeon room
'In this case, the map's unit of measure is in feet,
'and is only about 100 feet across.
'use the AddUserPoint method to be able to
' assign point to an object in feet.
oMap = GetCurrentMap()
oRoom = CreateMapObject( "Polygon")
oRoom.FillPattern = "Floor, gray"

' create a square room
oMap.AddUserPoint( oRoom, 10, 10)
oMap.AddUserPoint( oRoom, 10, 20)
oMap.AddUserPoint( oRoom, 20, 20)
oMap.AddUserPoint( oRoom, 20, 10)
oMap.AddObject( oRoom)
```

3.8.3.4 ApplyColorSchemeSelected

```
ApplyColorSchemeSelected( sName: string)
```

This method applies a Color Scheme (also known as a 'Complexion' in the Character Sketcher) to any currently selected Submap objects. Color Schemes are lists of color replacements associated with a name.

Example:

```
'Applies the 'Red Head' complexion to all selected Submap objects
'on a Character Sketcher sketch
oMap = GetCurrentMap()
oMap.ApplyColorSchemeSelected( "Red Head")
RefreshMap
```

3.8.3.5 BreakApartSelected

```
BreakApartSelected()
```

This method breaks apart all currently selected complex polygons (polypolygons) into their component polygons. This is the equivalent to the Actions-Combine Polygons-Break Apart menu item.

3.8.3.6 BringSelectedToFront

```
BringSelectedToFront()
```

Brings the currently selected objects to the front. This is the same funtionality as the Edit-Bring to Front menu item.

3.8.3.7 CanUndo

```
CanUndo(): boolean
```

Specifies true if the last action can be undone, and false if it can't be undone. A false value is typically returned when you back out of the last 5 actions.

3.8.3.8 ClearAll

```
ClearAll()
```

Removes all the objects from the map. Gone!

3.8.3.9 ClearFields

```
ClearFields()
```

Clears all of the map's custom fields.

3.8.3.10 ClearPaletteReplacements

```
ClearPaletteReplacements()
```

Clears any existing Palette Replacements that have been assigned to any object on the map.

3.8.3.11 ClearPaletteReplacementsSelected

```
ClearPaletteReplacementsSelected()
```

Clears any existing Palette Replacements that have been assigned to any currently selected object on the map.

3.8.3.12 ClearParents

```
ClearParents()
```

Removes all the parent maps (parent Fractal Mapper maps, FWE Maps, and graphic files) from the map.

3.8.3.13 ClearScript

```
ClearScript()
```

Clears (removes) any scripts assigned to the map (this is primarily map event handlers).

3.8.3.14 CloneSelected

```
CloneSelected( nType: integer)
```

Clones the currently selected objects based on *nType*. The possible values of *nType* are:

```
0 - Clone the object normally (make a copy of the object)
1 - Clone the object to a polygon.
2 - Clone the object to a polyline.
3 - Clone the object to a polypath.
```

3.8.3.15 CombineSelected

```
CombineSelected( mode: integer)
```

This combines the currently selected polygons into a PolyPolygon - a complex polygon with multiple parts. The mode parameter specifies how the polygons are combined:

```
0 - Add
1 - Subtract
```

```
   4 - Exclude (XOR)
```

See the application's documentation for information on what each of these methods do. These are the equivalent of the Actions-Combine Polygons menu items.

3.8.3.16 ConfigBool

```
ConfigBool( sAttrib: string, bVal: boolean)
```

Sets the attribute specified by *sAttrib* to the boolean value *bVal* on all the currently selected objects. For the list of valid attributes, see the ConfigureSelected() method.

3.8.3.17 ConfigInt

```
ConfigInt( sAttrib: string, nVal:integer)
```

Sets the attribute specified by *sAttrib* to the integer value *nVal* on all the currently selected objects. For the list of valid attributes, see the ConfigureSelected() method.

3.8.3.18 ConfigReal

```
ConfigReal( sAttrib: string, rVal: float)
```

Sets the attribute specified by *sAttrib* to the real (floating point) value *nVal* on all the currently selected objects. For the list of valid attributes, see the ConfigureSelected() method.

3.8.3.19 ConfigString

```
ConfigString( sAttrib: string, sVal: string)
```

Sets the attribute specified by *sAttrib* to the string value *sVal* on all the currently selected objects. For the list of valid attributes, see the ConfigureSelected() method.

3.8.3.20 ConfigureSelected

```
ConfigureSelected( sAttrib: string, nVal: integer, sVal: string, bVal: boolean, rVa
```

Sets the attribute specified by sAttrib to the appropriate value nVa, sVal, bVal, or rVal, depending on the data type of sAttrib. In most cases, it's more convenient to use one of the wrapper methods (ConfigBool(), ConfigReal(), etc).

The possible attribute values are listed below. The data type (integer, string, etc) that is used with the attribute is listed beside each attribute in parenthesis. They are (I) - Integer, (S) - String, (B) - Boolean, (R) - Float/Real.

```
Alpha (I)
ArrowEndStyle (I)
```

```
ArrowSize (I)
ArrowStartStyle (I)
BrushColor (I)
ClearFill (B)
EllipseSegmentRange (I)
EllipseSegmentStyle (I)
FillPattern (S)
FillPatternBrightness (I)
FillPatternScale (I)
FillStyle (I)
Flipped (B)
FontBold (B)
FontColor (I)
FontItalic (B)
FontName (S)
FontSize (I)
FontUnderline (B)
FractalFactor (I range 0-10)
GridAlpha (I)
GridColor (I)
GridFont (S)
GridFontSize (I)
GridHeight (I)
GridLabelAlign (I)
GridNumbering (B)
GridSync (B)
GridType (I)
GridWidth (I)
Layer (I)
LineScaled (B)
LineThickness (I)
LineType (I)
Locked (B)
MapLink (S)
MapLinkStyle (I)
Mirrored (B)
PenColor (I)
RandomSeed (N)
ReRender
Rendered
Rotation (I)
TextAlign (I)
Transparent (B)
ZoomCeiling (R)
ZoomFloor (R)
```

Example:

```
   'Set the fill color
   oMap = GetCurrentMap()
   oMap.ConfigureSelected( 'BrushColor', RGB( 200, 215,200), '', false, 0)  'only t
```

Example 2:

```
   'Same as above, using a wrapper method
   oMap = GetCurrentMap()
   oMap.ConfigInt( 'BrushColor', RGB( 200, 215,200))
```

3.8.3.21 CreatePolarArraySelected

```
CreatePolarArraySelected( nRings: integer, nCount: integer, nSpacing: integer, nDeg
```

Creates either a Polar or Stadium array from the currently selected objects. nRings is the number of rings in the array. nCount is the number of copies of the objects per ring. nSpacing is the distance, in the mapper's units, between rings. nDegStart and nDegEnd specify the arc of the array (use 0, 360 for a full circle). if bStadium is true, a Stadium array is created, otherwise a Polar array is created. See the application documentation for information about the two types of circular arrays.

3.8.3.22 CreateRectangleArraySelected

```
CreateRectangleArraySelected( nCols: integer, nRows: integer, nSpaceX: integer, nSp
```

Creates a Rectangle Array from the currently selected objects nCols horizontally and nRows vertically. The space between each object is given by nSpaceX and nSpaceY. This spacing is in the mapper's internal units, not the maps' user defined units (feet, miles, etc.).

3.8.3.23 CreateSubmap

```
CreateSubmap( sFile: string; Top, Left, Bottom, Right: integer)
```

Creates a new map file named *sFile*, out of the rectangle specified in world units by *Top*, *Left*, *Bottom*, and *Right*.

3.8.3.24 CreateSymbolFromSelected

```
CreateSymbolFromSelected( bReplace: boolean; sFile: string)
```

Creates a submap (ie, 'mapping symbol', 'facial feature') from the currently selected objects. if *bReplace* is true, the selected objects are removed from the map after the symbols is created. if *sFile* is anything other than an empty string (""), the new submap is immediately saved to that file.

3.8.3.25 DeleteObject

```
DeleteObject( n: integer)
```

Deletes the object at index n of the Map's array of MapObjects.

When cycling through the map's array of MapObjects, be sure to cycle **backwards**, since once an object is removed, all the objects higher in the array are shuffled down one. Failure to do this may mean skipping objects in a loop, or passing invalid index positions to other functions.

3.8.3.26 DeleteSelected

```
DeleteSelected()
```

Deletes the currently selected objects on the map.

3.8.3.27 ExplodeSymbolSelected

```
ExplodeSymbolSelected()
```

Breaks apart all selected vector mapping symbols into their component parts.

3.8.3.28 Fit

```
Fit( o: MapObject; w: float): boolean
```

Flushes the object o's nodes to any neighboring objects' nodes that are within the distance w (in world units). Return value is true if any changes to the object are made.

3.8.3.29 FitSelected

```
FitSelected( w: float)
```

Fits (flushes against) the nodes of all selected objects to any neighboring objects' nodes within the distance w (in world units). This is the same action that is performed when Actions - Flush Against is selected from the menu.

3.8.3.30 GetDefaultLayerName

```
GetDefaultLayerName( n: integer): string
```

Returns the *default* name of layer number n on the map. The default name is what Fractal Mapper calls layer n before it gets renamed by the user (if it gets renamed, that is).

3.8.3.31 GetField

```
GetField( s: string): string
```

Returns the value of the Map's custom field s. If no custom field by that name exists at the map level, an empty string is returned.

Example:

```
'iterates through all the objects, looking for the
'one with an object custom field called
''Last Modified', and sets the text of that
'object to the value of the 'Last Modified On'
'map level custom field.
```

```
Sub OnLoad()
   Dim s, n, i, o, f, oMap
   oMap = GetCurrentMap()
   s = oMap.GetField( "Last Modified On")
   n = oMap.ObjectCount()
   For i = 1 To n
      o = oMap.GetObject( i -1)
      f = o.GetField( "Last Modified")
      If Len( f) > 0 Then
         f.Text = s
      End If
   Next
End Sub
```

3.8.3.32 GetFieldByIndex

```
GetFieldByIndex( n: integer): string
```

Returns the value of the n-th custom field, or an empty string if there is no n-th custom field.

Example: (see the GetFieldCount() example)

3.8.3.33 GetFieldCount

```
GetFieldCount(): integer
```

Returns the number of custom fields that have been assigned to the map.

Example:

```
 'iterates through all the map level custom fields
Dim n, i, s, v, oMap
oMap = GetCurrentMap()
n = oMap.GetFieldCount()
For i = 1 To n
   s = oMap.GetFieldNameByIndex(i)
   v = oMap.GetFieldByIndex(i)
   ... Do something With the values...
Next
```

3.8.3.34 GetFieldNameByIndex

```
GetFieldNameByIndex( n: integer): string
```

Returns the name of the n-th custom field, or an emtpy string if there is no field at that index.

Example: (see the GetFieldCount() example)

3.8.3.35 GetFirstSelected

```
GetFirstSelected(): integer
```

Returns the index of the first selected object in the map's array of MapObjects.

3.8.3.36 GetLayerName

```
GetLayerName(n: integer): string
```

Returns the name of layer *n*.

3.8.3.37 GetNodeAtXY

```
GetNodeAtXY( x: integer, y: integer): integer
```

Returns the index of a selected object's node at point (x,y), or -1 if no node lies at that point.

3.8.3.38 GetObject

```
GetObject( n: integer): MapObject
```

Returns the MapObject at index *n* of the map's internal array of MapObjects.

3.8.3.39 GetObjectAtXY

```
GetObjectAtXY( x: integer, y: integer): integer
```

Returns the index of the MapObject in the map's internal array of MapObjects that resides (or includes) the point (x,y), or -1 if no object is at that point.

3.8.3.40 GetSelectedCount

```
GetSelectedCount(): integer
```

Returns the number of items that are currently selected.

3.8.3.41 GroupSelected

```
GroupSelected( bGroupOn: boolean)
```

Logically groups the currently selected MapObjects together. This is the same as the Object-Group menu item.

3.8.3.42 InitLayers

```
InitLayers()
```

Resets layer names back to their defaults, and sets all layers to visible and selectable.

3.8.3.43 IntersectionSnap

```
IntersectionSnap( o: MapObject, Range: float)
```

Snaps the first and last segments of the MapObject o to the nearest line segment intersection within the distance Range. This is the same functionality as the Intersection Snap.

3.8.3.44 InvertSelectedWithConstraints

```
InvertSelectedWithConstraints()
```

Inverts the currently selected objects - selects all currently unselected objects, and de-selects currently selected objects. This method takes into account layer settings (if they are selectable) and zoom windows.

3.8.3.45 IsRasterSymbolSelected

```
IsRasterSymbolSelected(): boolean
```

Returns true if any of the currently selected objects are raster symbols (png, bmp, etc).

3.8.3.46 isSelected

```
isSelected( n: integer): boolean
```

Returns true if the MapObject at index *n* of the Map's array of MapObjects is selected, and false if it is not.

3.8.3.47 isValidLayer

```
isValidLayer( n: integer): boolean
```

Returns true if *n* is a valid layer identifier. (i.e., if it's between 0 and 255)

3.8.3.48 LayerIsSecret

```
LayerIsSecret(n: integer): boolean
```

Returns true if layer n is flagged as secret.

3.8.3.49 LayerIsSelectable

```
LayerIsSelectable( n: integer): boolean
```

Returns true if the layer number n is set as selectable, and false if it isn't.

3.8.3.50 LayerIsVisible

```
LayerIsVisible(n: integer): boolean
```

Returns true if the layer number n is visible, false if it is hidden.

3.8.3.51 LoadFromFile

```
LoadFromFile( sFile: string): boolean
```

Clears the existing MapObjects from the map, and loads the map from the file named *sFile*. Returns false if a failure occurs.

3.8.3.52 MergeLines

```
MergeLines()
```

Merges the currently selected lines together, if possible. (i.e., if their start and end points match)

3.8.3.53 MirrorFlipSelected

```
MirrorFlipSelected( bMirror: boolean, bFlip: boolean)
```

Flips and/or mirrors the select objects along a common horizontal axis (flip) or vertical axis (mirror). If bMirror is true, the objects are mirrored. If bFlip is true, the objects are flipped.

3.8.3.54 Modified

```
Modified()
```

Saves the current state of the map to the undo cache, and flags the map as modified. Use the isModified 51 property to check the modified state of the map.

Example:

```
    'rerenders all the selected objects, but makes it 'undoable' by the user.
    oMap = GetCurrentMap()
    oMap.Modified()
    oMap.ConfigInt( 'Rerender', 0)
```

3.8.3.55 MoveBack

```
MoveBack()
```

Moves all selected objects back one increment in the front-back order.

3.8.3.56 MoveForward

```
MoveForward()
```

Brings all selected objects forward by one increment in the front-back order.

3.8.3.57 MoveSelected

```
MoveSelected( dx: integer, dy: integer)
```

Moves the currently selected objects *dx* units horizontally, and *dy* units vertically. *dx* and *dy* can be negative.

3.8.3.58 NewMap

```
NewMap()
```

Clears all the objects from the map, and frees all resources. Ready to start anew!

3.8.3.59 ObjectCount

```
ObjectCount(): integer
```

Returns the number of objects on the map. This values does not include all the objects found in submaps (mapping symbols).

3.8.3.60 PlaceSubmapAtScale

```
PlaceSubmapAtScale( sSubmap: string; x, y: integer; nDefaultDim: integer): MapObjec
```

This method places a Submap object, as given by it's file name, centered at position x, y. The Submap will be scaled according to the relationship between the Submap's size (in user defined units) and the Map's size (in user defined units). For example, if a symbol was defined as 10 feet across, and the PlaceSubmapAtScale() method was used to place the symbol on a map that's 100 feet across, the symbol would end up being 1/10th the width of the map, centered at point x,y (in the the map's internal coordinates). Units of measure will be converted where possible between English (U.S.) and Metric measurements by the Mapper. For example, you can place a 10 foot wide symbol on a map that's defined as 200 meters across, and it will be scaled correctly.

If the program cannot determine a scale to use (for example, if no unit of measurement is defined for the map) the symbol will be placed at a width of *nDefaultDim*, in world units.

If the file sSubmapfile does not exist the method returns null. Otherwise, the method returns the new Submap object.

Example:

```
sFile = "Tree.fmp"
oTree =  oMap.PlaceSubmapAtScale( sFile, x, y, 64)
```

3.8.3.61 ProximitySnap

```
ProximitySnap( o: MapObject, Range: float)
```

Snaps the first and last points of MapObject o to the closest node of any other object on the map within the a range of Range units. This is the same as the Proximity Snapping feature found on the Map menu..

3.8.3.62 ResizeMap

```
ResizeMap( NewWidth, NewHeight: float; NewUnit: string)
```

Resizes the map the map, enlarging or shrinking it according to the new size. NewWidth and NewHeight should be values in user units.

3.8.3.63 RotateSelected

```
RotateSelected( n: integer)
```

Rotates the currently selected objects by *n* degrees around a common center mass.

3.8.3.64 SaveToFile

```
SaveToFile( sFile: string): boolean
```

Saves the map to a file named *sFile* in the Mapper's own file format. Returns false if an error occurs.

3.8.3.65 SaveToGraphic

```
SaveToGraphic( sFile: string; nWidth: integer; Left: integer; Top: integer; Right:
```

This method saves a specified portion of the map to a graphics file. The file to be created is *sFile*. The graphic format created can be either bitmap, JPEG, or PNG, based on the file extension of *sFile*. If the file name has an extension of .jpg, a JPEG image is created. If .png, a PNG file is created. Otherwise the image is in a bitmap format. The width of the image to be created, in pixels, is specified by *nWidth*.

The height of the image is based on the rectangle being exported. For example, if the section exported is twice as high as it is wide, the height of an image would be 2 x *nWidth*.

The region of the map exported is defined by the four variables *Top*, *Left*, *Right*, *Bottom*. These are values defining a section of the map in the mapper's internal units. To export a section of the map by using the user defined measurement units (feet, meters, etc), first convert the measurements with the UserToWorld() method. To export the full map, pass 0,0, Map.Width, Map.Height as the rectangle values.

Example:

```
'exports a map to a 400 pixel wide bitmap file.
oMap = GetCurrentMap()
oMap.SaveToGraphic "c:\temp\mymap.bmp", 400, oMap.UserToWorld( 0), oMap.UserToWo
```

3.8.3.66 ScaleSelected

```
ScaleSelected( sx: float, sy: float)
```

Scales the currently selected MapObjects by the ratios *sx* horizontally, and *sy* vertically. Note that these are ratios, and not percentages. Thus, to double the size of a MapObject, you'd use the values (2,2) and not (200, 200).

Example:

```
'doubles the size of the selected MapObjects
oMap = GetCurrentMap()
oMap.ScaleSelected( 2, 2)
```

3.8.3.67 ScenarioBuilder

```
ScenarioBuilder( sFile: string; css: string; nPictureWidth, nDefaultPicWidth: integ
```

This method allows you to generate a Scenario Builder document from your map. The parameters are as follows:

- *sFile*: The name of the file to created. This must be given.
- *css*: The actual HTML CSS code to use. This is not a file name, but a string containing the actual CSS <style> codes to use.
- n*PictureWidth*: the width in pixels of the main map picture.
- *bMainPict*: if true, outputs the main map picture.
- *bPictLinks*: if true, generates a clickable set of hyperlinks off the main map picture to the corresponding object notes.
- *bIllustrations*: if true, individual object illustrations are placed into the document.
- *bBlockTextOnly*: if true, only block text entries are output. if false, both block text and notes are output.
- *bTOC*: if ture, a linked table of contents is generated.
- *bDefaultIllustrations*: if true, all objects with notes will have default illustrations generated for them automatically, if an illustration area is otherwise not selected for the object.

3.8.3.68 SelectAll

```
SelectAll( bSelected: boolean)
```

Sets the 'Selected' property of all objects on the map (regardless of layer settings or zoom windows) to *bSelected*.

3.8.3.69 SelectAllInRect

```
SelectAllInRect( x1: integer, y1: integer, x2: integer, y2: integer)
```

Selects all MapObjects that have nodes within a rectangle defined by (x1, y1, x2, y2). This function takes layer and zoom window settings into consideration.

3.8.3.70 SelectAllWithConstraints

```
SelectAllWithConstraints()
```

Selects all MapObjects on a map. This function takes layer settings and zoom windows into consideration. Thus, items on layers marked as 'unselectable' will not be selected by this method.

3.8.3.71 SelectAllWithText

```
SelectAllWithText( sText: string)
```

Selects all objects on the map that contain the text *sText*. This is the same as the Select by Example-Select Containing Text... menu item.

3.8.3.72 SelectGroup

```
SelectGroup( nGroupID: integer)
```

Selects all objects that belong to the group *nGroupID*. See the MapObject class's GroupID property.

3.8.3.73 SelectMatching

```
SelectMatching( sType: string)
```

This method performs Select by Example selection. It selects all objects on the map that share the attribute values of the currently selected objects based on *sType*. If more than one object is currently selected, then all objects on the map that share the same attribute value with any of the selected objects will be selected. For example, two objects on a map are currently selected - one a polygon, and the other a mapping symbol. If you call this method with 'BaseType' as the parameter, all polygons and all mapping symbols will be selected.

The valid values of *sType* are:

```
BaseType (note that all mapping symbols are considered to be of the same 'basety
BrushColor
FillPattern
PenColor
LineType
SymbolType
Layer
FontFace
FontSize
ZoomFloor
ZoomCeiling
```

3.8.3.74 SelectObject

```
SelectObject( n: integer, bSelected: boolean, BypassGroup: boolean)
```

Sets the Selected property of the object at index *n* of the map's array of MapObjects to *bSelected*. If *BypassGroup* is set to false, this applies this selection action to all members of the group to which this MapObject belongs, if any. If set to true, this method does not select the other members of the MapObject's group, if any.

3.8.3.75 SendSelectedToBack

```
SendSelectedToBack()
```

Sends the currently select MapObjects to the back. This is the same functionality as Edit-Send to Back .

3.8.3.76 SetBackgroundMap

```
SetBackgroundMap( sFile: string)
```

Sets the map's background map or image to *sFile*. *sFile* can be another Fractal Mapper map, a Fractal World Explorer map, or an image file (bmp, png, etc).

3.8.3.77 SetBevelSelected

```
SetBevelSelected( nBevelStyle: integer; nWidth: integer; nAngle: integer; intensity
```

Applies a bevel style to all selected objects.

nBevelStyle specifies the type of bevel to apply. The valid style values are:

```
0 - None
1 - Flat
2 - Rounded
3 - Sloped Up
4 - Stepped
5 - Small Steps
6 - Frame
```

nWidth represents the width, in pixels, of the bevel. Normal ranges are 1-100.

nAngle is the direction of the light source. Valid range is 0-360. (0 degrees is off to the right).

intensity is the intensity of the lighting. Valid range is 1-200

3.8.3.78 SetBlurSelected

```
SetBlurSelected( nBlurStyle: integer; nWidth: integer)
```

Applies a blur to all selected objects.

nBlurStyle is the type of blur to apply. The valid values are:

```
0 - None (removes the blur)
1 - Standard (fast bi-directional box blur)
2 - Feathered Edge
3 - Inverse Feather
```

nWidth is the width of the blur, in pixels. Normal range is 1 - 100.

3.8.3.79 SetColorMode

```
SetColorMode( nMode: integer)
```

Sets the color mode (color theme) for the map. The valid values for *nMode* are:

```
0 - Normal, Full Color
1 - Black (all colors are black.  For masks)
2 - Black and White, images in Grayscale
3 - Grayscale
4 - Sepia Tone
```

3.8.3.80 SetDropShadowSelected

```
SetDropShadowSelected( style: integer, color: integer, alpha: integer, x: integer,
```

Applies/Removes a vector drop shadow to/from the selected objects. style is one of the following drop shadow styles:

```
  0 - Drop Shadow Off
  1 - Drop Shadow On
```

Color is the color of the shadow. *Alpha* specifies the transparency level, from 0 (transparent) to 255 (solid) of the shadow. *X* and *y* are the shift of the shadow horizontally and vertically, in world units.

3.8.3.81 SetField

```
SetField( s: string, v: string)
```

Creates a custom field called *s*, and sets it's value to *v*. If a custom field by the same name already exists, it's value is replaced by *v*. To delete a field, set the value to an empty string.

Example:

```
Sub OnSave()
   ' sets some custom defined fields before saving
   oMap = GetCurrentMap()
   oMap.SetField "Last Modified On", CStr(Today)
   oMap.SetField "Created By", "Alan"
End Sub
```

3.8.3.82 SetGradientControlPointsSelected

```
SetGradientControlPointsSelected( x1: integer, y1: integer, x2: integer, y2: intege
```

Sets the gradient control points of all selected objects to points (x1,y1) and (x2, y2). The gradient control points are only used if the object is filled with a gradient fill.

3.8.3.83 SetGradientSelected

```
SetGradientSelected( c1: integer, a1: integer, c2: integer, a2: integer, style: int
```

Applies a gradient fill to the selected objects. *c1* and *c2* are colors - as returned by the RGB() function. *a1* and *a2* are integers from 0 to 255 indicating the alpha (transparency) value of each part of the gradient. 0 is transparent, 255 is solid. style denotes the type of gradient:

```
0 - None
1 - Linear
2 - Radial
```

3.8.3.84 SetLayer

```
SetLayer( n: integer; sName: string; bVisible: boolean; bSelectable: boolean; bSecr
```

Sets the attributes of layer number *n*. *sName* is the new name for the layer, *bVisible*, *bSelectable*, and *bSecret* set the visible, selectable, and secret attributes of the layer.

3.8.3.85 SetResolutionMode

```
SetResolutionMode( nMode: integer)
```

Sets the special effects DPI mode to *nMode*, and re-renders any objects that have special effects applied to them using the new mode. The valid values for *nMode* are:

```
0 - Low Resolution (map span represents 2000 pixels)
```

```
1 - Medium Resolution (map span represents 4000 pixels)
2 - High Resolution (map span represents 10000 pixels)
```

3.8.3.86 SetSelectedBrushColor

```
SetSelectedBrushColor(nColor: integer)
```

Sets the brush color of all selected objects to *nColor.*

3.8.3.87 SetSelectedPenColor

```
SetSelectedPenColor(nColor: integer)
```

Sets the pen/outline color of all selected objects to *nColor*.

3.8.3.88 SetShadowSelected

```
SetShadowSelected( nShadowStyle: integer; color: integer; nWidth: integer; nAngle:
```

Applies a special effects shadow to all selected objects. *nShadowStyle* specifies the type of shadow to apply. The valid values are:

```
0 - No Shadow (removes shadow)
1 - Drop Shadow
2 - Inner Shadow
```

Color is the color of the shadow.

nWidth is the width of the shadow's blur, in pixels. Typical range is 1-100.

nAngle is the angle, 0-360, of the light source (the shadow is cast in the opposite direction).

nOffset is the distance the shadow is shifted in the direction of *nAngle*. Typical range is 1-100.

3.8.3.89 SetTextureSelected

```
SetTextureSelected( sFile: string; nScale: integer; nIntensity: integer)
```

Sets the texture of all selected objects to *sFile*. *sFile* is the name of the grayscale image file being used as the texture. *nScale* is the scale at which to apply the texture. Typical values range 1-100. *nIntensity* is the intensity of the texture. Typical values range 1-100.

3.8.3.90 ShowAllLayers

```
ShowAllLayers()
```

Sets all layers to visible.

3.8.3.91 UnRasterRenderAll

```
UnRasterRenderAll()
```

Flags all objects that have special effects applied to them as unrendered, causing them to be re-rendered again during the drawing process.

3.8.3.92 UnRasterRenderSelected

```
UnRasterRenderSelected()
```

Flags all selected objects that have special effects applied to them as unrendered, causing them to be re-rendered again during the drawing process.

3.8.3.93 Undo

```
Undo()
```

Performs an 'Undo' operation against the map, if possible. This corresponds to the Edit-Undo menu item.

3.8.3.94 UpdateSymbolSpecHistory

```
UpdateSymbolSpecHistory(o: MapObject)
```

Updates the symbol memory settings for the symbol object, *o*.

3.8.3.95 UpdateSymbolSpecHistorySelected

```
UpdateSymbolSpecHistorySelected()
```

Updates the symbol memory settings for all selected objects.

3.8.3.96 UserToWorld

```
UserToWorld( n: float): float
```

Converts a distance from the user's unit of measure (feet, kilometers, etc) to the Mapper's internal measurement units.

3.8.3.97 WorldFromPixels

```
WorldFromPixels( n: float, z: float): float
```

Returns the distance, in the mapper's internal measurement units, represented by n number of pixels on your screen when the map is set to a zoom level of z.

3.8.3.98 WorldFromUnitDistance

```
WorldFromUnitDistance( n: float; sUnit: string): float
```

Returns the distance in world units represented by the distance *n* in measurement unit *sUnit*. If the current map does not use the same measurement unit as *sUnit*, the program will attempt to convert between the units where possible. If the program cannot convert between the units, 0 is returned.

Example:

```
oMap = GetCurrentMap()
n = oMap.WorldFromUnitDistance( 200, 'Feet')
```

3.8.3.99 WorldToUser

```
WorldToUser( n: float): float
```

Returns a distance in the user's unit of measurement from a value n given in the Mapper's unit of measurement.

3.9 InspirationPad Class

The InspirationPad class makes the Inspiration Pad Pro random generator engine available to you in scripts. The InspirationPad class allows you to open existing .ipt files, assign variables and definitions, and run the generators.

This is not to be confused with the Inspiration Pad feature of Fractal Mapper (the one that allows you to drag and drop random names onto the map). That is a simplified interface to the InspirationPad engine, and is controlled by a separate set of Inspiration Pad functions 18.

3.9.1 Related Functions

3.9.1.1 CreateIPadTable

```
CreateIPadTable( sFile: string): InspirationPad
```

Loads the .ipt file named *sFile* and returns an InspirationPad object for that generator if successful.

3.9.2 Properties

3.9.2.1 LastResultCount

```
LastResultCount(): Integer
```

Returns the number of results from the last call to the Go() method. This is usually the number of repetitions passed to Go(), but may be less if the table restricts the number of results.

3.9.3 Methods

3.9.3.1 AddVariableDef

```
AddVariableDef( sName: String; sVal: String)
```

Adds a variable named *sName* to the generator, and sets its value to *sVal*.

This is similar to the Inspiration Pad Pro *Set* command:

```
Set: Variable=Value
```

3.9.3.2 AddDefinition

```
AddDefinition( sName: String; sVal: String)
```

Adds a constant named *sName* to the current generator, and sets its value to *sVal*.

This is similar to the Inspiration Pad Pro *Define* command:

```
Define: constant=expression
```

3.9.3.3 ClearAllVars

```
ClearAllVars()
```

Clears all active variables in the generator.

3.9.3.4 ShuffleAll

```
ShuffleAll()
```

Shuffles tables to reset them for deck picks.

3.9.3.5 Go

```
Go( n: Integer)
```

Runs the generator *n* times. Use GetResult() to access the results of each iteration.

Example:

```
'Create a new IPad table object by loading the file named sPath
ipad = CreateIPadTable( sPath)

'set a variable to be accessed by the IPad table.
'you only need to add a variable definition if you need to pass
'some value from a script into an IPad table.
ipad.AddDefinition "somevar", "{1d6}"

'run the table 4 times
ipad.Go 4

'loop through and get each individual results
For i = 1 To 4
       s = ipad.GetResult( i-1)
       MsgBox s
Next
FreeObject ipad
```

3.9.3.6 GetResult

```
GetResult( n: integer): String
```

Returns result *n*, after using Go() to execute the generator.

3.10 Graphical User Interface Classes

3.10.1 Dialog Window

3.10.1.1 Related Functions

3.10.1.1.1 NewDialogWindow

```
NewDialogWindow(): DialogWindow
```

Creates a new dialog (modal) window, and returns the object reference.

```
Set w = NewDialogWindow()
 w.Top = 10
 w.Left = 20
 w.Width = 600
 w.Height = 600
 w.Caption = "My Window"
 w.Centered = True
```

3.10.1.2 Properties

3.10.1.2.1 Caption

```
Caption: String
```

This is the caption of the DialogWindow, and is what is displayed in the window's title bar.

3.10.1.2.2 Centered

```
Centered: Boolean
```

If *true*, the window is centered on the screen, regardless of what the Top and Left properties are set to.

3.10.1.2.3 Height

```
Height: Integer
```

This is the height of the DialogWindow object on the screen, in pixels. The window's Height value includes the height of the window's title bar as drawn by the operating system.

3.10.1.2.4 Left

```
Left: Integer
```

This is the Left position of the DialogWindow object on the screen, in pixels.

3.10.1.2.5 Top

```
Top: Integer
```

This is the top position of the DialogWindow object on the screen, in pixels.

3.10.1.2.6 Width

```
Width: Integer
```

This is the width of the window object on the screen, in pixels.

3.10.1.3 Methods

3.10.1.3.1 AddCheckbox

```
AddCheckbox(): Widget
```

Creates a Check Box widget and places it on the window. Return value is the Widget object created.

3.10.1.3.2 AddColorSelector

```
AddColorSelector(): Widget
```

Creates a Color Selector widget and adds it to the window. A Color Selector widget is a colored button that when clicked on displays a color selection window. Return value is the Widget object created.

Example:

```
'Sets the fill color of all selected objects to a selected color
m = GetCurrentMap()

'find the first selected object and get its brush color
n = m.GetFirstSelected()
If n > -1 Then
    o = m.GetObject( n)
    cl = o.BrushColor
Else
    cl = RGB( 255, 255, 255)
End If

Set w = NewDialogWindow()
 w.Top = 10
 w.Left = 20
 w.Width = 240
 w.Height = 240
 w.Caption = "Color Tester"
```

```
 w.Centered = True

cr = w.AddColorSelector()
 cr.SetPosition 20, 20, 100, 100
 cr.Color = cl

If w.ShowModal Then
    m.ConfigInt "BrushColor", cr.Color
End If
```

3.10.1.3.3 AddCombo

```
AddCombo(): Widget
```

Creates a Combo Box widget (an editable drop down list) and places it on the window. Return value is the Widget object created.

3.10.1.3.4 AddDropList

```
AddDropList(): Widget
```

Creates a fixed Drop Down List widget and places it on the window. Return value is the Widget object created.

3.10.1.3.5 AddFileOpen

```
AddFileOpen(): Widget
```

Creates a File Open widget and places it on the window. This widget is a combination of a Text Box and a button that lets the user select a file to open. Return value is the Widget object created.

Example:

```
fo = w.AddFileOpen()
 fo.SetPosition 320, 350, 250, 24
 fo.FileMask = "Fractal Mapper Maps (*.fmp)|*.fmp|All Files (*.*)|*.*"
 fo.FileName = "c:\temp\some.csv"
```

3.10.1.3.6 AddFileSave

```
AddFileSave(): Widget
```

Creates a File Save widget and places it on the window. This widget is a combination of a Text Box and a button that lets the user select a file to save as. Return value is the Widget object created.

3.10.1.3.7 AddGroupBox

```
AddGroupBox(): Widget
```

Creates a Group Box widget and places it on the window. Return value is the Widget object created.

3.10.1.3.8 AddImage

```
AddImage(): Widget
```

Creates an Image widget and places it on the window. Return value is the Widget object created.

3.10.1.3.9 AddLabel

```
AddLabel(): Widget
```

Creates a Label widget and places it on the window. Return value is the Widget object created.

3.10.1.3.10 AddListbox

```
AddListbox(): Widget
```

Creates a List Box widget and places it on the window. Return value is the Widget object created.

3.10.1.3.11 AddMemo

```
AddMemo(): Widget
```

Creates a multi-line Text Box widget and places it on the window. Return value is the Widget object created.

3.10.1.3.12 AddRadioGroup

```
AddRadioGroup(): Widget
```

Creates a Radio Button Group widget and places it on the window. Return value is the Widget object created.

3.10.1.3.13 AddTextEdit

```
AddTextEdit(): Widget
```

Creates a Text Edit (Text box) widget and places it on the window. Return value is the Widget object created.

3.10.1.3.14 AddTrackBar

```
AddTrackBar(): Widget
```

Creates a Track Bar widget (also known as a slider) and places it on the window. Return value is the Widget object created.

Example:

```
trk = w.AddTrackbar()
 trk.SetPosition 270, 480, 270, 50
 trk.MaxValue = 100
 trk.Position = 90
```

3.10.1.3.15 ShowModal

```
ShowModal(): Boolean
```

Displays the DialogWindow object. Returns *true* if the Ok button is pressed, *false* if the Cancel button is pressed.

3.10.2 Widget

3.10.2.1 Properties

3.10.2.1.1 Caption

```
Caption: String
```

This is the label for a widget. Not to be confused with a widget's *Text* property which holds a user editable text value.

3.10.2.1.2 Checked

```
Checked: Boolean
```

For Checkbox widgets, specifies if the Checkbox is checked or not.

3.10.2.1.3 Color

```
Color: Integer
```

For ColorSelector widgets, this is the currently selected color.

3.10.2.1.4 Columns

```
Columns: Integer
```

For Radio Groups, this is the number of columns to use when displaying options.

3.10.2.1.5 FileMask

```
FileMask: String
```

For FileOpen and FileSave widgets, this is the file mask thats used to filter file names. This is stored as a pipe-delimited string, as such: "description|mask|description|mask".

Example:

```
'To offer filters for .sector files, or files of any type:
fo.FileMask = "Maps (*.fmp)|*.fmp|All Files (*.*)|*.*"
```

3.10.2.1.6 FileName

```
FileName: String
```

For FileSave and FileOpen type widgets, this is the currently selected file. There is no guarantee that this is a valid file or location, so scripts should check the file's validity prior to accessing it.

3.10.2.1.7 Fit

```
Fit: Boolean
```

Applies to Image widgets. If *true*, the image is scaled to fit the size of the widget. If false, it is not (and is cropped if it is too big).

3.10.2.1.8 Height

```
Height: Integer
```

The height of the widget, as displayed on the window, in pixels.

3.10.2.1.9 ImageFileName

```
ImageFileName: String
```

For Image widgets, this is the name of the image file to display.

3.10.2.1.10 Left

```
Left: Integer
```

The left position of the Widget on the window, in pixels.

3.10.2.1.11 MaxValue

```
MaxValue: Integer
```

This is the maximum value of a TrackBar widget.

3.10.2.1.12 MinValue

```
MinValue: Integer
```

This is the minimum value of a TrackBar widget.

3.10.2.1.13 Position

```
Position: Integer
```

This is the current position of a TrackBar widget as set by the user.

3.10.2.1.14 Text

```
Text: String
```

This is the contents of a Text Edit or Memo's text box. The Text property is the user-editable text value of a widget. This is different from the *Caption*, which is the text used to label certain widgets.

3.10.2.1.15 Top

```
Top: Integer
```

The Top position of the Widget on the window, in pixels.

3.10.2.1.16 Width

```
Width: Integer
```

The width of the widget, as displayed on the window, in pixels.

3.10.2.2 Methods

3.10.2.2.1 AddItem

```
AddItem(s: String)
```

Adds an item to a ListBox, DropList, or ComboBox type widget.

Example:

```
lb = w.AddListBox()
 lb.Top = 300
 lb.left = 20
 lb.Width = 200
 lb.Height = 120
 lb.AddItem "Mountain"
 lb.AddItem "Dungeon"
 lb.AddItem "Ocean"
 lb.AddItem "Desert"
 lb.Text = "Dungeon"  'sets the selected item
```

3.10.2.2.2 SetPosition

```
SetPosition( x: Integer; y: Integer; w: Integer; h: Integer)
```

Sets the widget's position on the window to a Left value of *x*, Top value of *y*, Width of *w*, and Height of *h*. This is the same as setting the Left, Top, Width, and Height values individually.

3.11 Persistent Storage

The GoblinAPI provides several functions to support application wide persistent storage. This allows scripts to store and recall settings between uses. In addition, since this is a persistent storage system, the values stored using these functions are retained between uses of the program.

Data is stored by Name Spaces and Keys. A Name Space is a unique string which identifies the script or set of scripts using the set of data to be saved or recalled. This allows multiple scripts to use the same key names, as long as the Name Space is different. Keys are the name of the piece of data

being stored.

The advantage of using these built-in functions is that all reading and writing to the data files is handled for you, and the data is automatically stored in the correct application data directory for the user.

You should not use the persistent storage functions as a replacement for script variables, or for storing massive amounts of data. Rather, use these functions to store and recall limited amount of data (script settings, for example) in between uses of the script. Storing very large amounts of data will degrade the performance of these functions.

3.11.1 ClearPropertyStorage

```
ClearPropertyStorage()
```

Clears *all* values for *all* name spaces stored within the persistent storage system. This deletes all data stored by any script that has used the persistent storage system. When the function is called, the user will be prompted if they wish to continue the deletion.

3.11.2 RecallBoolean

```
RecallBoolean( sNamespace: string; sKey: string; bDefault: boolean): boolean
```

Retrieves a boolean value (true or false) from the persistent storage system.

sNameSpace is the name space within which the requested key resides.

sKey is the name of the key you are requesting.

nDefault is a default value to return if *sKey* does not exist.

Example: (see StoreString)

3.11.3 RecallNumber

```
RecallNumber( sNamespace: string; sKey: string; nDefault: float): float
```

Retrieves a number from the persistent storage system.

sNameSpace is the name space within which the requested key resides.

sKey is the name of the key you are requesting.

nDefault is a default value to return if *sKey* does not exist.

Example: (see StoreString)

3.11.4 RecallString

```
RecallString( sNamespace: string; sKey: string; sDefault: string): string
```

Retrieves a string/text value from the persistent storage system.

sNameSpace is the name space within which the requested key resides.

sKey is the name of the key you are requesting.

sDefault is a default value to return if *sKey* does not exist.

Example: (see StoreString)

3.11.5 StoreBoolean

```
StoreBoolean( sNamespace: string; sKey: string; bVal: boolean)
```

The StoreBoolean function stores a boolean value (true or false) within the persistent storage system. Data stored using the persistent storage system can be recalled between uses of a script, or even between uses of the program.

sNameSpace is any text that uniquely identifies this script or set of scripts. Since persistent storage is shared among all scripts, a unique namespace allows for the same key name to be used by many scripts without interference with each other.

sKey is the name of the key under which to store the value. To retrieve this value, you'd request the value that corresponds to this key.

bVal is the boolean value (true or false) to store.

Example: (see StoreString)

3.11.6 StoreNumber

```
StoreNumber( sNamespace: string; sKey: string; nVal: float)
```

The StoreNumber function stores a number within the persistent storage system. Data stored using the persistent storage system can be recalled between uses of a script, or even between uses of the program.

sNameSpace is any text that uniquely identifies this script or set of scripts. Since persistent storage is shared among all scripts, a unique namespace allows for the same key name to be used by many scripts without interference with each other.

sKey is the name of the key under which to store the value. To retrieve this value, you'd request the value that corresponds to this key.

nVal is the number to store.

Example: (see StoreString)

3.11.7 StoreString

```
StoreString( sNamespace: string; sKey: string; sVal: string)
```

The StoreString function stores a text value within the persistent storage system. Data stored using the persistent storage system can be recalled between uses of a script, or even between uses of the program.

sNameSpace is any text that uniquely identifies this script or set of scripts. Since persistent storage is

shared among all scripts, a unique namespace allows for the same key name to be used by many scripts without interference with each other.

sKey is the name of the key under which to store the value. To retrieve this value, you'd request the value that corresponds to this key.

sVal is the string/text value to store.

Example:

```
' store the text "castle" under the key named "building-type"

StoreString  'myscript', 'building-type', 'castle'

... other script activity, or in another run of the script ...

' then later, recall it
BuildingType = RecallString( 'myscript', 'building-type', '')
```