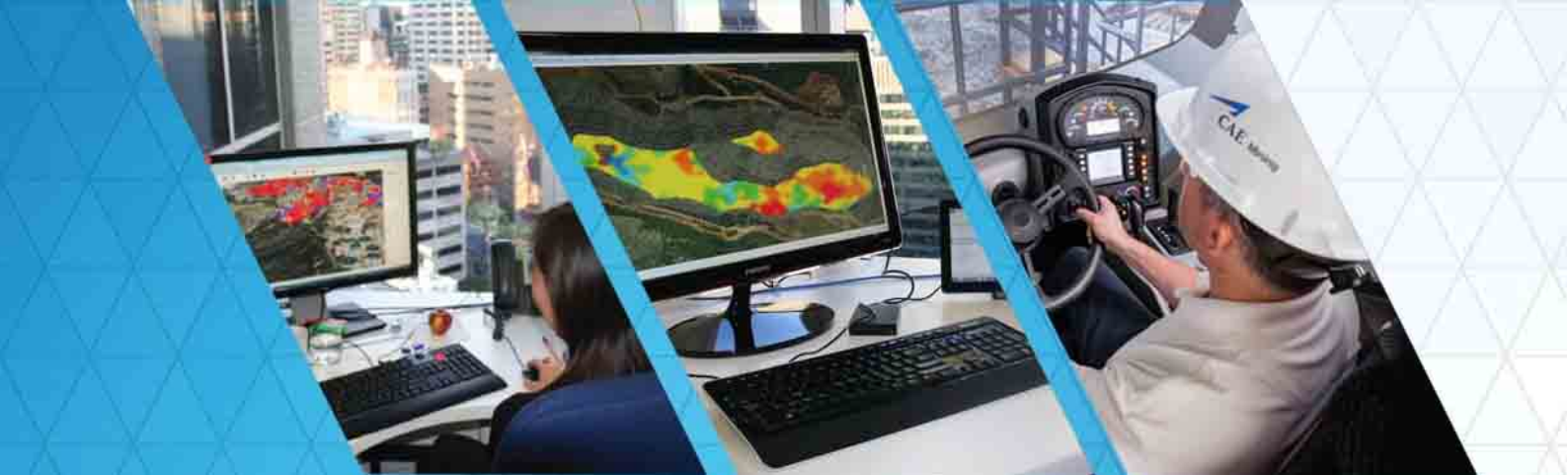


Studio 3



Scripting & Automation User Guide



This documentation is confidential and may not be disclosed to third parties without the prior written permission of CAE Mining Corporate Limited.

Contents

1	Overview	1
	Purpose of this document	1
	Prerequisites	1
	Acronyms and Abbreviations	1
	More information	1
2	Introduction	2
	The Customization Control Bar	2
	Capabilities and Possibilities	3
	Scripting as an Aid to Integration	3
	Files Referenced in this Guide	3
3	Recording and Replaying Scripts	5
	Before you Start	5
	Recording a Script	5
	Replaying a Script	9
	Automation vs. Interactivity	9
4	Viewing Your Script	10
	Viewing and Associating Help files	11
	Saving a Reference Copy of Your Script	12
	Running a Script from Internet Explorer	12
5	Simple Windows Methods	13
6	Example: Creating a Polygon	18
	Introduction	18
	Creating the Polygon Interface	18
	Adding the Polygon Script	22
	Running the Polygon Script	26
7	Example: Browsing and Copying	29
	ActiveX Controls	29
	Creating the Browse and Copy Interface	30
	Adding the Browse and Copy Script	33
	Further Browse and Copy Extensions	34
8	Example: Running a Macro	38
	Using XRUN to Execute the Macro	38
	Creating the macro interface	39
	Adding the Macro Script	42
	Running the Macro Script	47
9	More Examples	49
	Overview	49
	Studio 3 File Browser Example	49
	Model Prototype Script	51
	Accessing Records and Fields	53
	ExecuteCommand vs. ParseCommand	54

EXECUTIVE SUMMARY

Studio 3 provides a new, industry standard interface that allows you to write scripts using **JavaScript** or **VBScript**, or any COM-aware scripting language. These scripts can be embedded into an HTML document, which can be loaded into the Studio 3 Customization window to execute commands.

As well as running individual commands from scripts you can also run your existing macros and 'Command Line' programs (macros) which has always been a powerful feature of CAE Mining systems. You can even launch macros from scripts, getting the best of both worlds.

Scripting allows you to customize the operation of Studio 3 commands, and access the versatile suite of functions using an automation interface. The level of customization that you can perform will, of course, depend on the task or tasks you wish to undertake, and one of the key aspects of scripting is that your creations can evolve over time. In this sense, you become your own Developer, and Studio 3 provides the components and interfaces needed to set up your processes, super-processes, interfaces and even entire applications.

Due to the nature of the subject matter, a guide explaining 'how to script' needs to be pitched somewhere between providing background information to scripting, and attempting to explain every macro or script command available. This document intends to provide you with a sensibly-balanced introduction to the benefits of scripting in Studio 3, and also the opportunity to learn about fundamental elements of scripting both in a general context, and from within the confines of Studio 3. This document also gives you a chance to go through some practical exercises to give you a taster of what can be achieved.

Hopefully, this Guide will raise your awareness of the potential of automating some of your business processes using the wide variety of tools and components on offer, and provide a solid basis for learning more about this powerful aspect of Studio 3.

At CAE Mining, we are aware that the creation of scripts, particularly for complex solutions, takes time, and the layers of complexity and versatility possible can prove daunting. It is important that whatever scripts you create are robust and reliable, efficient, scalable and easy to instigate and use. The team at CAE Mining have expert knowledge of all CAE Mining packages, and provide a custom solutions service perfectly matched to your business requirements. If you wish to discuss the benefits of automated solutions for your Business, please contact your local CAE Mining Support Representative.

1 OVERVIEW

Purpose of this document

This document aims to provide:

- background information covering the basics of scripted solutions
- an understanding of how scripts are created, recorded and edited
- practical exercises to enhance your understanding of how scripts are constructed
- conceptual and specific information on *methods*, *properties* and *events*
- examples of how Studio 3 forms and other components can be accessed using scripts
- information on the options available for running macros

Prerequisites

This guide serves as a useful document if you are new to scripting with CAE Mining products, or want to learn more about how this particular aspect of CAE Mining software has evolved. However, it is highly recommended that you approach this document with the following skills to hand:

- It is recommended that you have some knowledge of established scripting methodologies, such as JavaScript and/or VBScript.
- You should have an appreciation of the object-oriented approach to programming, and understand the concept of objects, methods, properties and events.
- You should have an understanding of common Studio 3 commands and processes, and the Studio 3 interface.
- You have access to the Studio 3 Demo Data Set, as installed with each release of Studio 3. For more information, see *Files Referenced in this Guide*.
- You have **Microsoft FrontPage 2003**® installed on your PC with **Microsoft Script Editor**®, and have some familiarity with using these packages, although instructions will be given.

Acronyms and Abbreviations

The following Acronyms and Abbreviations are used throughout this document:

Acronym	Description
JS	JavaScript
VBS	VBScript
UI	User Interface
HTML	Hypertext Markup Language
MSE	Microsoft Script Editor
COM	Component Object Model

More information

- *JavaScript: The Definitive Guide*, by David Flanagan, Published by O'Reilly and Associates, 1998. ISBN: 1-56592-392-8
- *JavaScript Bible*: by Danny Goodman, Brendan Eich, Published by IDG Books Worldwide, ISBN: 0-76453-188-3

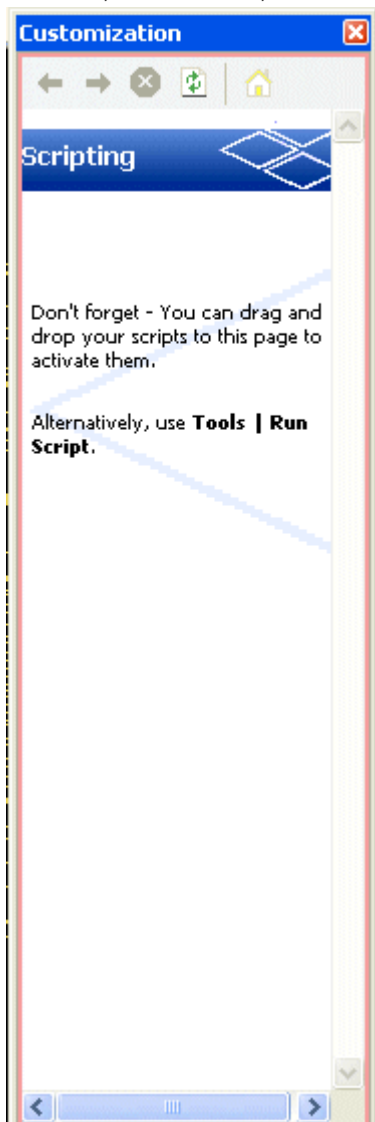
2 INTRODUCTION

Scripts can be written to provide rich and versatile user interfaces using HTML, with HTML being used to contain scripts, that in turn can access an extensive collection of tools and functions available with Studio 3 components.

For anyone already familiar with CAE Mining product scripting, this new facility can be thought of as a replacement for the **!SCREEN** macro command, but is many times more powerful, and flexible.

In its most simple form, you can use the scripting facilities of Studio 3 to record your own actions in script form, by selecting **Tools | Scripting | Start Recording** from the **Tools** menu. This creates a simple HTML document with **Execute** and **Help** buttons that will allow you to replay the sequence of commands you have previously recorded.

However, in the longer term, you will probably want to do more than that; you may want to change an interface so that, for example, you can specify both input and output file names; you may also need to specify field and parameter values, and then add extra buttons, radio buttons, text boxes, check boxes, and so on, in order to make your interface more flexible and provide a richer, more functional user experience.



Studio 3 provides a variety of options for the recording of scripts, and provides scriptable components to allow you to define and build your own solutions with the minimum of effort. The Studio 3 Application model, for instance, allows you to access a variety of highly useful methods and properties, such as the ability to open a file, access project data, run commands and processes, provide your own applications using HTML form components.

As scripts are constructed as HTML pages, anyone with some familiarity of web-based scripting techniques, and a basic appreciation of the Document Object Model (DOM) and Component Object Model (COM) technologies can get to grips with Studio 3 scripting quickly and easily.

The Customization Control Bar

One method of running scripts in Studio 3 is available via the **Customization** control bar. This bar can be viewed, if not already part of your Studio 3 profile, by selecting **View | Customization | Control Bars | Customization**.

This area of your system is used to both view and run scripted commands, and is in effect an HTML interface for the display of your own forms. A default form is created each time a script is recorded using the in-built script recording functions of Studio 3, but this can act as a starting point for more complex and useful scripted interfaces. The exercises in this document take you through the creation of such form interfaces (for the creation of a polygon, a 'browse and copy' file interface, and an interface that will allow you launch a macro file).

As the **Customization** control bar is an HTML interface, the

right-click menu shown will be the same as that shown for Internet Explorer, hence, script source can be edited and saved without having to exit your Studio 3 application.

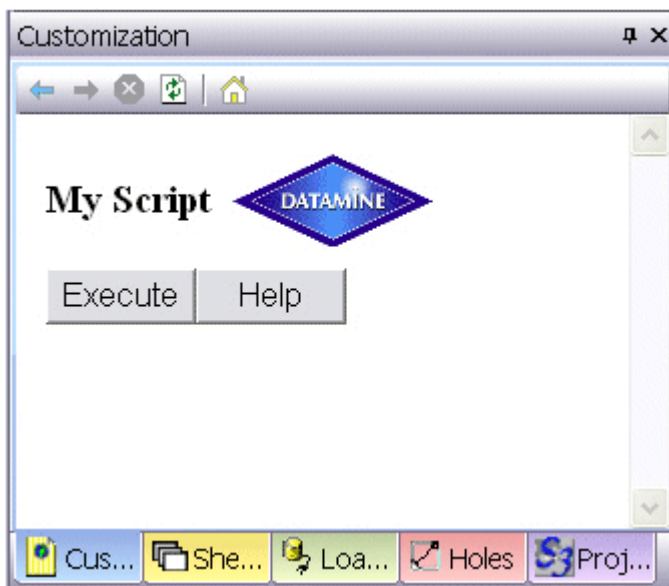
The figure on the right shows a typical recorded script in the **Customization** pane of Studio 3.

Changing the default interface is done by editing the HTML script. User interface design involves more than the addition of a text box or push button; you will also need to define an action or event that occurs when you enter data into a text box or click a button.

The 'functional' side of scripting, that is, what actually happens when an event is triggered, is achieved using JavaScript or VB Script.

Capabilities and Possibilities

You can run both Process and Design-Window commands from these scripts as well as carrying out the usual programming functions. You can also interface with other systems through scripting. As HTML, JavaScript and VBScript are well-grounded methodologies, Studio 3 scriptable components can be manipulated using a combination of Studio 3 'native' commands and other generic functionality. This document shows examples of both approaches. In fact, some Studio 3 functions were written with scripting in mind, and Studio 3 provides useful JavaScript functions to simplify tasks such as running macros and setting up pick lists, for example.



This guide takes you through steps ranging from recording and replaying scripts to designing your own interface.

The aim of the guide is *not* to teach you HTML and JavaScript/VBScript. These are both third-party languages and there are plenty of books and other resources available that can teach you from whichever level of experience you currently have (see the *More Information* section in the Introduction for some recommendations). Instead, this guide concentrates on the tools that are needed to create and manage HTML pages, macros and other files

that are used in Studio 3, and how to access and use these tools. For this reason, one of the prerequisites of this guide is an appreciation of web scripting languages, such as JavaScript.

Scripting as an Aid to Integration

Experienced users will recognize that the existence of an industry-standard COM interface to Studio 3 commands opens up many possibilities for integrating Studio 3 into their company's overall IT system. Studio 3 commands can now be accessed and executed remotely from just about any programming language or system, including **Access Basic**, **Sybase PowerBuilder**, **Borland Delphi**, **Visual Basic** or **Visual C++**.

Files Referenced in this Guide

There are three types of supporting file installed with your Studio 3 package:

- **Demo script files:** these HTM files contain examples of scripts that are created with the exercises described.
- **Macro files:** macro (menu) files created by following the exercises.
- **Datamine files:** supporting Studio 3 data files that are utilised within the exercises.

In all cases, if a particular exercise relies on data either prepared by previous exercises, or if it is essential to go through a list of instructions previously, an indication will be made of the file(s) or steps required. However, the following file types can be found in the directories listed below, with a standard installation of Studio 3:

- Demo **Script** files can be found at
C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts.
- Demo **Macro** files can be found at
C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts.
- Demo files can be found at
C:\Database\DMTutorials\Data\VBOP\Datamine.

3 RECORDING AND REPLAYING SCRIPTS

In this section, you are going to be introduced to the Scripting functionality within Studio 3 by recording a sequence of Studio Processes. You will then replay this Script.

The recording of scripts using Studio 3's proprietary functions is an important and useful aspect of scripting, as it will allow you to create the necessary code (if necessary, for future editing) using the Studio 3 user interface, tools and dialogs.

Before you Start

The exercises referred to in this section rely on the following tasks being performed first:

- You have created a new project within Studio 3, labelled 'Scripting User Guide'. For more information on creating projects, follow the exercise *Creating a New Project* in the *Working with Projects* topic of the *Project Files, Data Types and Objects* session of the **Studio 3 Introductory Tutorial**.

The project Location should be:

C:\Database\DMTutorials\Projects\S3ScriptTut\ProjFiles

- Either add the following files to the project using the exercise *Adding Files to a Project* described in the *Working with Projects* topic of the *Project Files, Data Types and Objects* session of the **Studio 3 Introductory Tutorial**:
 - *dhCollar.dm* - Drillhole collars file.
 - *dhAssays.dm* - Drillhole assays file.
 - *dhLith.dm* - Drillhole lithology file.
 - *dhSurvey.dm* - Drillhole survey file.
- OR;
- Import the files using the exercises outlined in the *Importing Text Data* topic of the *Data Importing* session of the **Studio 3 Introductory Tutorial**.

If you do not have access to these files, please contact your CAE Mining Support Representative.

Recording a Script

In the following exercise, you are going to record and save a script, run a sequence of commands and stop the script from recording, using the following procedure. This script will utilize the **SORTX**, **JOIN**, **HOLMER** and **DDLST** commands to create a desurveyed drillhole file from specified component files:

1. Select **Tools | Scripting | Start Recording**.
2. In the **Save As** dialog, you should navigate to your current project directory, and define the new script name as 'Holes3D' and then click **Save**. Note that the default extension for a script file is *.html.

3. Studio 3 is now ready to record your sequence of processes and other commands, indicated by the message in the **Command** window with the text "Commands are now being recorded".
4. Type 'sortx' in the **Command** window and press <Enter>. The **SORTX** command sorts any file into ascending (or optionally descending) order of the keyfields.
5. In the **SORTX** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN	dhcollars
	Output Files OUT	tempcollars
<i>Fields</i>	Fields KEY1	BHID

6. You are now going to repeat step 4 - type 'sortx' in the **Command** window and press <Enter>.
7. In the **SORTX** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN	dhassays
	Output Files OUT	tempassays
<i>Fields</i>	Fields KEY1	BHID
	Fields KEY2	FROM

8. Again, repeat step 4.
9. In the **SORTX** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN	dhlith
	Output Files OUT	templith
<i>Fields</i>	Fields KEY1	BHID
	Fields KEY2	FROM

10. Again, repeat step 4.
11. In the **SORTX** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN	Dhsurvey
	Output Files OUT	tempsurvey
<i>Fields</i>	Fields KEY1	BHID
	Fields KEY2	AT

12. Instead of repeating the **SORTX** command, this time you are going to run the JOIN process, which sorts any file into ascending (or optionally descending) order of the keyfields. Type 'join' into the Command line and press <Enter>.

13. In the **JOIN** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN1	tempcollars
	Input Files IN2	tempassays
	Output Files OUT	temp1
<i>Fields</i>	Fields KEY1	BHID
<i>Parameters</i>	SUBSETR	1

14. You are now going to run the **HOLMER** command, which merges two sets of drillhole data samples with different downhole distances. Type 'holmer' into the Command line and press <Enter>.

15. In the **HOLMER** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN1	temp1
	Input Files IN2	templith
	Output Files OUT	temp2
<i>Fields</i>	Fields BHID	BHID
	Fields FROM	FROM
	Fields TO	TO

16. Type 'desurv' in the Command line and press <Enter>.

17. In the **DESURV** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN1	temp2
	Input Files IN2	tempsurvey
	Output Files OUT	dholes

18. Type 'ddlist' in the Command window and press <Enter>.

19. In the **DDLST** dialog define the settings as shown below and click **OK**:

Tab Name	Field Name	Value
<i>Files</i>	Input Files IN1	dholes

20. Stop the script recording by selecting **Tools | Scripting | Stop Recording** from the **Tools** menu.

21. In the **Command** window, ensure that the following has been output by the 'ddlist' command:

```

Command
-----
Filename      Tutorials\DMTutorials\Projects\S3IntroTut\ProjFiles\MyProj1\holes.
dm
EP FILE CREATED USING DESURVEY ON 05/11/02
-----
FILE CONTAINS      1045 RECORDS EACH OF LENGTH      37
-----
FIELD  TYPE WORD.NO  STORED  START      DEFAULT
-----
BHID   A      1      Y      1
BHID   A      2      Y      2
BHID   A      3      Y      3
BHID   A      4      Y      4
X      N      1      Y      5      0.0
Y      N      1      Y      6      0.0
Z      N      1      Y      7      0.0
LENGTH N      1      Y      8      0.0
AO     N      1      Y      9      0.0
BO     N      1      Y     10     90.0
CO     N      1      Y     11     0.0
FROM   N      1      Y     12
TO     N      1      Y     13
XPT    N      1      Y     14     0.0
YPT    N      1      Y     15     0.0
ZPT    N      1      Y     16     0.0
ENDDEPTH N     1      Y     17
REFSYS A      1      Y     18
REFSYS A      2      Y     19
REFSYS A      3      Y     20
REFSYS A      4      Y     21
REFMETH A     1      Y     22
REFMETH A     2      Y     23
REFMETH A     3      Y     24
ENDDATE A     1      Y     25
ENDDATE A     2      Y     26
ENDDATE A     3      Y     27
ENDDATE A     4      Y     28
AU     N      1      Y     29
CU     N      1      Y     30
DENSITY N     1      Y     31
LITH   A     1      Y     32
LITH   A     2      Y     33
LITH   A     3      Y     34
LITH   A     4      Y     35
NLITH  N     1      Y     36
RADIUS N     1      N     0      1.0
-----
>>> DDLIST Complete <<<

```

For those of you already familiar with CAE Mining Studio processes, you may recognize that the above set of procedures is already combined into a single, existing process – also called **HOLES3D**. This command can be run from the Studio 3 command line, and performs all of the above steps (after you have specified all necessary fields) plus additional validation before creating a desurveyed drillhole file.

Your version of the Holes3D command has been saved as an HTML file, that can be edited (this guide shows you how to do this, in the following section). When you have created your script, you can replay it by issuing a single command.

Replaying a Script

In this exercise, you are going to replay a sequence of recorded commands as a script:

1. Clear the **Command** window by right clicking in the window and selecting **Clear** from the popup menu.
2. Select **Tools | Scripting | Run Script...**
3. In the **Browsing for script file...** dialog, browse and find the script named 'Holes3D' and select **Open**.
4. The script will be loaded into the **Customization** window of Studio 3. Contained in the window is the name of the script (in this case *Holes3D*) and two buttons; **Execute** and **Help**.
5. Run the script by clicking **Execute**.
6. Examine the **Command** window output and verify that it is the same as before.

Automation vs. Interactivity

Note that your *HOLES3D* script (this will be referred to in italics to differentiate it from the current Studio 3 **HOLES3D** process) requires no additional input from anyone running it; you have specified all of the files, fields and parameters necessary to run the command. Whilst being highly-automated, this does have the disadvantage of being relatively inflexible as a command, as it can only ever generate one set of results, and create one downhole file, from one set of source files.

In later exercises, you will learn how to allow input to be provided 'at run time' so that your script can process the values entered into an HTML form interface. The provision of an interface (or 'form') to capture user input before processing commands is an essential part of scripting if you wish to create interactive processes. Forms not only provide a prompt for users to specify required parameters for your scripts, they also provide a valuable resource for attaching different sections of code to different areas. For example, you may want to issue an alert prompt if someone has forgotten to supply a certain parameter, or you may wish to report that a command may take a few minutes to complete, on clicking a form's control.

4 VIEWING YOUR SCRIPT

When creating a script, you are, in fact, creating a web document containing a combination of JavaScript-based commands and standard HTML syntax. You can edit this script using whichever tools you are familiar with, but for the purpose of this guide, you should use FrontPage 2003 and Microsoft's Script Editor (for information on how to configure these products, see Chapter 3 – *Configuring FrontPage*).

In this section, as it is intended to provide a quick overview of the source code generated by the Studio 3 scripting function, we will be viewing the data in Notepad.



In order to view the source, with the *Holes3D* script loaded in the **Customization** window (see *Replaying a Script* in the previous Chapter), right-click the background of the script somewhere where plain text exists (e.g. not an image) and select **View Source** from the menu and the script *Holes3D* will then be loaded into **Notepad** and displayed.

FrontPage 2003 and the Microsoft Script Editor will be providing the tools you need to edit this file. However, as you are only going to have a quick look at the file now to get an appreciation of the general structure of the HTML file that is created, you can continue with **Notepad**.

The initial block of HTML code (between the <HEAD> and <SCRIPT ID...> tags) contains header information, and generally only requires editing if you are going to be creating your own Javascript functions. You will be doing just this later in the guide. The next three lines declare some local variables that will be used within the script. These variables are declared as a **DMApplication** object, to provide access to Studio 3 commands and processes (**oDmApp**), a scripting object, otherwise known as the 'script helper' (**oScript**) and a browser object (**oDMBrowser**).

Following the variable declaration are three main functions that are crucial for the effective running of Studio 3 scripts. These functions are:

- **AutoConnect**: this function creates instances of objects declared as **oDmApp** and **oScript** which allow the connection to the Studio 3 application and script helper respectively.
- **btnExecute_onclick**: this runs the sequence of commands that were recorded when the **Execute** button is clicked. **btnExecute_onclick** makes a call to **AutoConnect**.
- **btnHelp_onclick**: this runs the help file script when the **Help** button is clicked. Finally, at the end, you can find the HTML code describing the interface. For all recorded commands, this initially includes the CAE Mining logo and the position and size of the **Help** and **Execute** buttons on the HTML page. This information is held within an HTML form, for example, the code for the two default buttons and image is as follows:

```
<table border="0" cellpadding="5" cellspacing="0">
```

```

<tr>
<td><b>tester</b></td>
<td></td>
</tr>
<tr>
<td align="centre" colspan="2"></td>
</tr>
<tr>
<td align="centre" colspan="2"><input type="button" value="Execute"
name="btnExecute" language="javascript" onclick="return
btnExecute_onclick()" style="width: 75"><input type="button"
value="Help" name="btnHelp" language="javascript" onclick="return
btnHelp_onclick()" style="width: 75"></td>
</tr>
</table>

```

The CAE Mining logo is shown on the default page, as referenced by the IMG SRC syntax shown. The Execute and Help button form items each have an action associated with their respective onclick events; **btnExecute_onclick** and **btnHelp_onclick**. These represent two of the functions mentioned in the bulleted list on the previous page.

This default interface is provided as a starting point, and can (and most likely, will) be edited extensively during the creation of custom scripts.



The full qualified path to the image *dmlogo.gif* is shown in the above example. However, other company logos can be displayed by changing the script to reflect the location of the new logo.

Viewing and Associating Help files

By default, the **Help** button for each created script links to the general Scripting Help file. This file provides a brief description of Studio 3's scripting facility. However it may be useful for you to provide a script-specific file, accessible from the script interface. To do this, you can use the default HTML for the Help button, but make a slight change the associated function, or you can create a new function for your own file path.

The Help Path

If you view a recorded script's source code (providing it hasn't been edited already), you will see the `btnHelp_onclick` function in the <HEAD> section of the page, as described in the previous section.

This function creates a new window object, and specifies a URL built up from a standard path. In the default code, the new window displays a topic in the Studio 3 compiled help file (CHM).

To understand how it operates, the following information provides a section-by-section breakdown of the function itself:

First, the function is declared:

```
function btnHelp_onclick() {
```

The next variable sets up a comma-delimited list of parameters, as expected by the new window **open** method, called at the end of the function:

```
var features = "height=600,width=400,status=yes,toolbar=no,menubar=no,location=no";
```

The next line declares a variable defining the title of the new window, which will again be passed into the **open** method:

```
var common = "YourScriptName";
```

A file path is then constructed, based on the standard installation path for Studio 3 components, and to ensure DOS-compliance, all spaces are replaced with line space codes (%20):

```
var installpath = "C:/Program Files (x86)/CAE/Studio/";  
installpath = installpath.replace(" ", "%20");
```

Next, the variable *helpcommand* is declared, consisting of the standard CHM access prefix sequence and the installation path variable, followed by the location of the Studio 3 scripting overview HTML page.

```
var helpcommand = "mk:@MSITStore:" + installpath +  
"/Help/DatamineStudio.chm::/Studio_3_General/Concept_Studio%20%20Scripting%20Overview.  
htm";
```

To create your own *helpcommand*, you can declare it here, in full, and omit the install path declaration and build-up.

Finally, the new window is created, and the CHM contents are displayed.

```
window.open(helpcommand , common, features);  
}
```

Saving a Reference Copy of Your Script

Hopefully you were successful in creating your own copy of *Holes3D.htm*. However if you want to view the reference copy and/or save a copy, you can do so in the following way:

1. Select the **Project Files** control bar in Studio 3 and open the *All Files* section. For more information on using the **Project Files** control bar, please refer to your online Help.
2. Double-click the entry *Holes3D* or *_scr_Holes3D* to open a copy into a window in the main graphical area with the title *Command Automatically recorded html script*.
3. Right-click and use **View Source** from the menu.
4. Save the file from the **Notepad** application by using **File | Save As...** from the menu. You should use the **Save As** dialog to browse to the location of the project and provide a new name for the file (make sure you add the '.htm' extension to the file name and don't use the default of '.txt').

This method of saving the reference copy of a file is used frequently in this tutorial, so it is important to understand it.

Running a Script from Internet Explorer

Instead of saving a copy of the script and running it from the **Customization** Window in Studio 3 you can run the script directly from the **Internet Explorer** window.

You must already have Studio 3 loaded, and it is recommended that you run the script from within Studio 3 (as opposed to a browser) as this will give you immediate access to all the other commands, plus it is easier to display both the script and the other windows simultaneously.

5 SIMPLE WINDOWS METHODS

This section of the User Guide aims to show you how in-built Windows methods can be used in conjunction with scripted Studio 3 commands, providing generic and Studio 3-specific functionality.

There are three commonly used windows methods which can be added to a script.

- **alert()** - displays a message to the user.
- **confirm()** - prompts the user to click an **OK** or **Cancel** button to confirm or cancel an operation.
- **prompt()** - prompts the user to enter a text string or some values.

All these methods display a simple dialog to the screen called a popup dialog box.

There are 3 exercises in this section, each of which will describe how each method outlined above can be used.

Before you Start

The exercises referred to in this section rely on the following tasks being performed first:

- A copy of the file *_scr_SortCollars.htm* is used in the exercises below. This script file, part of the Studio 3 demo data set, will be able to sort collars and outputs the file *tempcollar.htm*, and can be found at **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts**.

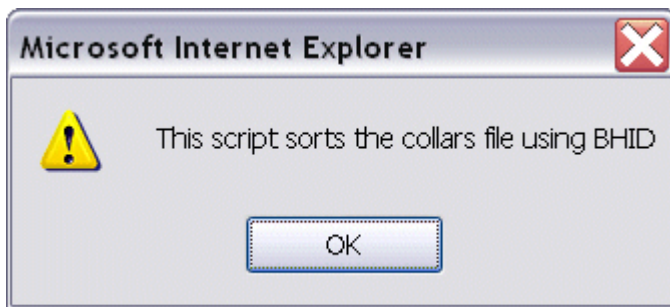
Adding Alert

In this exercise we will add an alert message to a script. The **alert()** method is used to display a dialog box containing the value of a variable or a text string. For example insert an **alert()** at the start of the function to display a title.

1. Start Microsoft FrontPage 2003®.
2. Use the **File | Open...** menu command and using the **Open File** dialog browse to the file *_scr_SortCollars.htm* located in the **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts** directory
3. Click **Open**.
4. Locate the line starting with **oDmApp.ParseCommand**.
5. Update the **alert** command with suitable text as shown below:

```
alert("This script sorts the collars file using BHID");  
oDmApp.ParseCommand("sortx &IN=dhcollar &OUT=SortedCollars *KEY1=BHID  
@BINS=5 @ORDER=1");
```

6. Use the **Find** dialog (**Edit | Find...** from the menu) to search for '_scr_SortCollars' text, click the **Find** button. When the text is found remove the '_scr_' characters from the file. There should only be two occurrences of this text.
7. Save the file using **File | Save As...** from the menu and in the **Save As** dialog enter the name 'SortCollars.htm' and click **Save**. The file extension should be defined for you. Leave this application open with the loaded file as we shall use this in later exercises.
8. Select **Tools | Scripting | Run Script...** from the main menu.
9. In the **Browsing for script file...** dialog select the *SortCollars.htm* file (this file was created in step 6).
10. In the **Customization** window of Studio 3, note that the script is loaded and shows the 'SortCollars' text next to the CAE Mining logo.
11. Clear the **Command** window by right-clicking in the window and selecting **Clear** from the menu.
12. Execute the script.
13. The popup message is displayed:



14. Click **OK**.
15. The **Command** window now shows the output from the script.

Adding Prompts

The **prompt()** method displays a dialog which includes a message that you set and provides a text field for you to enter a response. Two buttons; **Cancel** and **OK**, give you the option to:

- **Cancel**: cancel the entire operation
- **OK**: accept the input and continue the operation

The **prompt()** method has two arguments; the first is the prompt message which is displayed on the dialog. The second argument is a default reply which is displayed in the text box. If you don't want to specify a default reply, enter two successive double quotes ("").

The method returns a value when you click on one of the buttons. Clicking **Cancel** returns a value of null, irrespective of what is in the text box. Clicking **OK** returns the value of the text string in the text box.

We will use the **prompt()** method to request the name of the collars file, and store the value in variable *cfile*. You should then add a test to check the value of *cfile*. If *cfile* is empty ("") or null then return will terminate the function.

Finally, you will need to replace the hard-coded file name *dhcollar* by the variable name *cfile* in the command which sorts the file. Note that *cfile* is a variable, and so must not be included within the double quotes. Therefore, the argument for **oDmApp.ParseCommand** is split into three parts separated by a '+' sign as shown in the example below. Make sure there is a space immediately in front of **&OUT=tempcollars**, so that files, fields and parameters are all separated by spaces.

The following exercise takes you through the steps necessary to achieve this;

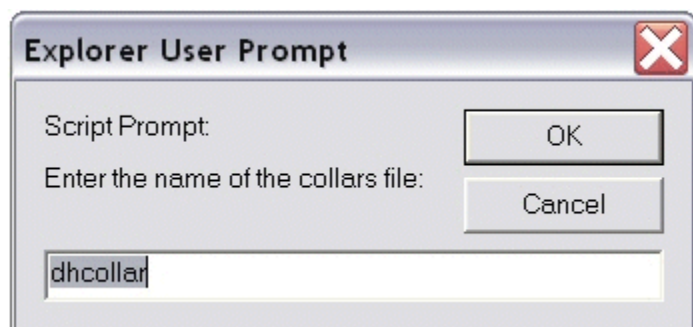
1. Continue to use the *SortCollars* file in FrontPage 2003.
2. Enter the following two lines between the alert and the oDmApp command as shown below:

```
alert("This script sorts the collars file using BHID");  
var cfile = prompt("Enter the name of the collars file: ", "dhcollar");  
if (cfile == "" || cfile == null) return;  
oDmApp.ParseCommand("sortx &IN=" + cfile + " &OUT=SortedCollars  
*KEY1=BHID @BINS=5 @ORDER=1");
```



In the text above; the '=' is an assignment and the '==' is a comparison. The '||' is a boolean operator meaning OR. The whole expression reads 'if cfile is empty or cfile is null then return'. It should be noted that if the '=' replaces the '==' then an assignment would take place within the 'if' decision, however, the code would not be incorrect. This is a common mistake using this language construct.

3. Click the **Save** icon or use the **File | Save** menu command.
4. Select **Tools | Scripting | Run Script...** from the main menu.
5. In the **Browsing for script file...** dialog select the *SortCollars.htm* file created in Exercise 1.
6. Clear the **Command** window by right clicking in the window and selecting **Clear** from the menu.
7. Execute the script. Note that the alert message still shows. Click **OK**.
8. The prompt message is now displayed and shows the default *dhcollar* file name, as shown by the image to the right.
9. Clear the text box and click **OK**. Note that nothing appears in the Command window output as the



filename was blank.

- Repeat step 7 and with the *dhcollar* filename in the text box click **OK**. Note that you can quickly reload a saved script by right-clicking the browser window and selecting **Refresh**.
- The **Command** window now shows the output from the script.
- Perform this exercise for the output file as well.

The text after the edit for the output file should look similar to the following. The names of the variables and text used may differ:



```
alert("This script sorts the collars file using BHID");
var cfile = prompt("Enter the name of the collars file: ",
"dhcollar");
if (cfile == "" || cfile == null) return;
var ofile = prompt("Enter the name of the output file: ",
"SortedCollars");
if (ofile == "" || ofile == null) return;
oDmApp.ParseCommand("sortx &IN=" + cfile + " &OUT=" + ofile + "
*KEY1=BHID @BINS=5 @ORDER=1");
```

Adding Confirm

The **confirm()** method displays a dialog that includes a user specified message and provides two buttons, **Cancel** and **OK**. This is similar to **prompt()** except that the only response is to click one of the two buttons. The method returns the Boolean value true if **OK** is clicked or false if **Cancel** is clicked. You can then add code to act according to which button has been clicked. For example use **confirm()** to display the names of the files that have been selected, and to abort the operation if **Cancel** is clicked.

The following exercises describes how you would add a Confirm prompt:

- Continue to use the *SortCollars* file in FrontPage 2003.
- Enter the following confirm line between the comparison for the output file and the **oDmApp** command as shown below:

```
alert("This script sorts the collars file using BHID");
var cfile = prompt("Enter the name of the collars file: ", "dhcollar");
if (cfile == "" || cfile == null) return;
var ofile = prompt("Enter the name of the output file: ",
"SortedCollars");
if (ofile == "" || ofile == null) return;
if (!confirm("The input file selected is: \n" + "Collars: " + cfile
+"\n\n The output file is: " + ofile)) return ;
oDmApp.ParseCommand("sortx &IN=" + cfile + " &OUT=" + ofile + "
*KEY1=BHID @BINS=5 @ORDER=1");
```

The '\n' character above forces a new line in the output text.



The '!' in front of **confirm()** reverses the Boolean value. For example if **Cancel** is clicked, the value of **confirm()** will be false. However adding '!' will make the response true, so the action will be return and the operation will terminate. The whole expression reads 'if the files are not **OK** then return'.

3. Click the **Save** icon or use the **File | Save** menu command.
4. Select **Tools | Scripting | Run Script...** from the main menu.
5. In the **Browsing for script file...** dialog select the *SortCollars.htm* file created in exercise 1.
6. Clear the **Command** window by right-clicking in the window and selecting **Clear** from the menu.
7. Execute the script. Note that the alert message still shows as well as both prompts. Click **OK** to all prompt and alert dialogs.
8. The confirm message is now displayed, as shown on the right.



The **alert()** and **confirm()** methods are particularly useful for debugging a script. Although the **prompt()** method can be used to enter a value it is a rather crude way of doing so. One of the main advantages of scripting is using HTML to create an attractive interface, as you will discuss in the following sections.

6 EXAMPLE: CREATING A POLYGON

Introduction

This section introduces many of the ideas behind scripting, and develops a useful example, step by step. You will be developing a new interface with text boxes and buttons, and adding the script. From now on you will be using Microsoft FrontPage 2003® and the Microsoft Script Editor (MSE)® as described in Chapter 3. Make sure they are both installed before continuing.

Studio 3 contains a handy process called **CONPOL**, for finding the convex hull around a set of points. The script developed in this tutorial integrates the **CONPOL** process with the **Design** Window, so that any selection of the point and string data currently in the **Design** Window can be passed to **CONPOL**, and the resulting polygon loaded back into the **Design** Window. The script processes X and Y coordinates, i.e. it works in a plan view.

The user may either create some point data in the **Design** window or use the specially created file for the example.

The creation of an interface, in isolation, is only one part of a series of steps necessary for creating useful, interactive, scripted functionality. This Chapter takes you through a set of exercises that will help you understand these areas, and examples of how they can be achieved.

- First, you will create the interface using HTML form commands.
- Then, you will add the script that 'sits behind' the interface.
- Finally, you will run the script and see the results.

As you become more familiar with scripting, it is likely that you will start to adopt your own method of working; for example, you may prefer to produce all of the 'behind the scenes' scripts before designing and implementing an interface, or create a library of useful scripts that you can use for a multitude of tasks, and minimize the amount of duplication you would otherwise have to endure when creating scripts performing similar actions.

Onto the exercises:

Creating the Polygon Interface

At the bottom of the FrontPage 2003 window are four tabs: *Design*, *Split*, *Code* and *Preview*.

Extension Distance	<input type="text" value="0"/>
Filter	<input type="text"/>
	<input checked="" type="checkbox"/> Include point data
	<input type="checkbox"/> Include string data
<input type="button" value="OK"/>	<input type="button" value="Cancel"/>

The *Design* or *Split* tab is where you will design the HTML form, and the *Preview* tab is where you can preview what you have designed, and even operate the controls and commands. You use the *Code* tab only when you are adding the script to the page. The interface you are going to create is shown on the left. It consists of a table, some buttons, check boxes, text and text boxes.

In the following exercises, you are going to:

- Draw a table to contain the interface objects
- Add some text and text boxes
- Add check boxes for data selection
- Add **OK** and Cancel buttons
- Finishing and tidying up the interface
- Assigning names and values to the controls

Before you Start

The exercises referred to in this section rely on the following tasks being performed first:

- The file `_vb_scr_points.dm` will be used to test the script. This file is located (with a standard installation) at `C:\Database\DMTutorials\Data\VBOP\Datamine` and should be loaded into the Project and displayed in the **Design** window.

Drawing a table to contain the interface objects

You will use the *Design* tab to insert a table into the interface:

1. Start Microsoft FrontPage 2003.
2. Create a new blank page using **File | New...** from the menu and selecting *Blank Page* from the **New Page** window.
3. Save the file using **File | Save As...** from the menu and in the Save As dialog enter the name 'Polygon.htm' and locate the file in the **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts** directory. Click **Save**. The file extension should be defined for you. Note that partial HTML sections have already been added to the file.
4. Select **Table | Insert | Table...** from the menu. In the **Insert Table** dialog increase the value of *Rows* to '5' and click **OK**. This will have inserted a table containing five rows and two columns into the *Design* tab.



Instead of **Table | Insert | Table...**, you can use **Table | Draw Table...** and actually draw the lines separating the rows and two columns. This feature is particularly useful if you want to draw tables with highly complex arrangements of cells.

Adding some text and text boxes

Next you will add some text and text boxes to the table. The text describing the contents of the text boxes will be placed into the left hand cells of the table and the text boxes themselves into the right hand cells.

1. Position the cursor within the top left hand cell of the table and type 'Extension Distance'.

2. In the next row down, type 'Filter'.
3. Position the cursor in the top right hand cell and select **Insert | Form | Text Box** from the menu. This will place a text box within the cell.
4. Now repeat step three and insert another text box in the cell below.

You may prefer to have the forms elements available on a toolbar. To do this, select **Insert | Form**, then drag the sub-menu away to turn it into a toolbar as shown below.



Adding Check boxes for data selection

Next you will add two check boxes to the table. The text describing these will be associated with the check box itself:

1. Position the cursor in row three column two, directly below the text box.
2. Select **Insert | Form | Checkbox** from the menu. With the cursor in the cell, type in the text 'Include point data' to its right.
3. Repeat step two by selecting the next cell down and typing in the text 'Include string data' to it right.

Finishing and tidying up the interface

Now, you can improve the look of the table. This step is not absolutely necessary, but it will improve the visual integration with Studio 3, and will make the interface behave better if the script window is resized.

All of the table layout improvements can be done with right-clicks of the mouse, and changing values in the resulting dialogs. However, this lesson will give the menu command alternatives.

1. Select all the cells in the table by positioning the cursor in the top left cell and with the left mouse button depressed drag and select all the cells in the table.
2. Select **Table | Table Properties | Cell** from the menu. In the **Cell Properties** dialog, clear the *Specify width* and *Specify height* check boxes to allow the browser to select the best fit width and height for the cells. Click **OK**.
3. Place the cursor inside the table and use **Table | Select | Table** from the menu. This will select the whole table (it is different from selecting the cells as can be seen below).
4. Select **Table | Table Properties | Table** from the menu. In the **Table Properties** dialog change the *Layout Alignment* to *Center* and clear the *Layout Specify width* and *Specify height* check boxes. click **OK**. This has centered the table within the *Design* tab and will keep it there as the script window is resized.

In step four:



- Setting the alignment to *Center* will keep the table in the center of the script window as it is resized.
- Deselecting the minimum width and height will allow the browser to re-size the table to give the smallest possible size.
- Within this dialog, setting the border size to zero will hide the lines between the columns and rows, but the table structure will keep things in their proper places. We have chosen to leave the borders in place for this example.

5. Highlight the bottom row of cells (those containing the buttons) and select **Table | Table Properties | Cell** from the menu. In the **Cell Properties** dialog, change the *Layout Horizontal* alignment to *Center*, to center the buttons in the cells. Click **Apply** and then **OK**.
6. To give the HTML page a name; right-click the *Design* tab, and select **Page Properties** from the popup menu. In the **Page Properties** dialog enter the text 'Polygon Script' into the *Title* field. Click **OK**.
7. Save the interface by selecting **File | Save** from the menu.

Assigning names and values to the controls

Before you can make anything happen, we need to assign names and initial values to all of the interface controls. Names can be anything that you wish, but remember that upper and lower case letters are distinct (the name 'this' is different from 'This'). A good idea is to prefix the names of controls with two or three letters that identify the type of control, e.g. **btnOK** for an **OK** button, or **tbSelectedFile** for a text box that will contain the name of a selected file.

There are a few ways to assign a name to a control we shall use either the right click method on the control and use the **Form Field Properties...** from the popup menu. Alternatively, you could just double click the control with the left mouse. Either of these methods will display the relevant properties dialog.

1. Double-click on the *Extension Distance* text box in the *Design* tab. In the **Text Box Properties** dialog type 'tbExtension' into the *Name* field and type '0' (zero value) into the *Initial value* field. This will initialise the text box with data when it is displayed. Click **OK** to accept the changes.
2. Double-click the *Filter* text box in the *Design* tab. In the **Text Box Properties** dialog type 'tbFilter' into the *Name* field. Click **OK** to accept the changes.
3. Double-click the *Include point data* check box in the *Design* tab (you may have to click on the actual box that represents the check box itself rather than the text). In the **Check Box Properties** dialog type 'cbIncludePoints' into the *Name* field. Select the *Initial state* option. Click **OK** to accept the changes.
4. Double-click the *Include string data* check box in the *Design* tab (you may have to click on the actual box that represents the check box itself rather than the text). In the **Check Box Properties** dialog type 'cbIncludeStrings' into the *Name* field. Select the *Not checked* option for the *Initial state*. Click **OK** to accept the changes.

5. Double-click the **OK** Button in the *Design* tab. In the **Push Button Properties** dialog type 'btnOK' into the *Name* field and type 'OK' in the *value/label* field. Click **OK** to accept the changes.
6. In order to make the **OK** button the same size as the **Cancel** button, adjust the *value/label* field text to insert spaces before and after the word 'OK'.
7. Double-click the **Cancel** Button in the *Design* tab. In the **Push Button Properties** dialog type 'btnCancel' into the *Name* field and type ' Cancel' in the *value/label* field. Click **OK** to accept the changes.
8. Save the interface by selecting **File | Save** from the menu.
9. Click on the *Preview* tab to see how the interface will actually look.
10. Click on the *Code* tab and look at the values that have been entered into the HTML code for you by designing the interface.



JavaScript is notoriously 'fussy' when it comes to typing syntax; be sure to use the exact names for the controls as given in the text above as they are case-sensitive.

Adding the Polygon Script

The first step is now complete: you now have an HTML user interface, together with all the hooks for the script.

The script language used is not a part of Studio 3 - you can use any of a number of standard scripting languages: **JavaScript**, **VBScript**, **PerlScript**, PowerBuilder's **PowerScript**, etc. The two languages supported by FrontPage 2003 are VBScript and JavaScript.

Unlike macros and menus, scripts are usually made up of a number of event handlers. These are small sections of script that are executed by the browser when some relevant event occurs.

Only two events are handled in this script:

- When the window loads (a window **onload** event occurs). We use this event to make a connection to the **Studio Application Object**. This is, in effect, our portal to Studio 3 functionality.
- When the **OK** button is pressed (a button **onClick** event occurs). We use this event to ensure that point and/or string data is processed and saved.

In this section, you will be going through the following exercises:

- Setting up MSE
- Adding the window's **onload** event
- Adding the **OK** button's **onClick** handler

Before you start

The exercises referred to in this section assume the following tasks have been undertaken already:

- Ensure Microsoft Script Editor is installed and integrated with Microsoft FrontPage 2003®. In FrontPage 2003® you should find the menu item under **Tools | Macro | Microsoft Script Editor**. The menu looks similar to the one shown below:



- We will continue to use the *Polygon.htm* file created in the previous exercise.

Setting up MSE

To start entering the script that makes the HTML come alive, first click on the Code tab, then select Tools | Macro | Microsoft Script Editor from the menu. A new window appears, showing your interface in HTML. We will use this application to define code sections for button click and window events.

1. To configure the Script Editor to use JavaScript, select **View | Property Pages** from the menu.
2. In the **DOCUMENT Property Pages** dialog that is displayed, on the *General* tab, set the *Client* field to 'JavaScript'. Click **OK**. You will need to perform this task every time the Script Editor is used on a different document. The Script Editor does not save JavaScript as the default.
3. Select **View | Toolbox** from the menu. This will display a list of objects such as buttons and check boxes etc.
4. Select **View | Other Windows | Document Outline** from the menu. This will display the list of objects that have been added to the interface.

Adding the windows onload event

In this exercise we will add an event that will be fired when the window is loaded.

1. Above the main window (the one displaying the code), you will notice two drop down list boxes. The left hand one contains the text 'Client Objects & Events' and the right hand one contains the text '(No Events)'. Using the left hand drop-down list, select the item [window].
2. Using the right hand drop down box select the item [onload]. The following text has been added to the code in the main window. If after selecting the [onload] item, the words 'Sub window_onload()' appears in the code window, Script Editor is configured to use VBScript.

```
function window_onload() {  
}
```

3. We now need to enter the JavaScript code to be executed when this event occurs. First, we need to establish a connection between the script and the **CAE Mining**

Studio Application Object. To accomplish this we must define a Datamine application variable, and refer to a routine to instantiate the object.

4. Before the function **window_onload()** type the following text

```
var oDmApp = null;  
var oScript = null;
```

5. Within the body of the **window_onload()** function (between the '{' and '}' characters), type the text 'AutoConnect()'. This instructs the **onload** event to call a function **AutoConnect** that we will now write. This routine will use the variable **oDmApp** defined in step 5.
6. After the **window_onload()** function and after the '}' symbol copy and paste the following text:

```
function AutoConnect() {  
    oScript = new ActiveXObject("DatamineStudio.ScriptHelper");  
    oScript.initialize(window);  
    oDmApp = oScript.getApplication();  
    if (oDmApp== null)  
        return false;  
    else  
        return true;  
}
```

7. The final code should look like the following:

```
var oDmApp = null;  
var oScript = null;  
function window_onload() {  
    AutoConnect();  
}function AutoConnect() {  
    oScript = new ActiveXObject("DatamineStudio.ScriptHelper");  
    oScript.initialize(window);  
    oDmApp = oScript.getApplication();  
    if (oDmApp== null)  
        return false;  
    else  
        return true;  
}
```



The variables **oDmApp**, **oScript** and the **AutoConnect** function are necessary for every scripted solution that connects to the **Studio Application Object**. The **AutoConnect** function initialises the **Studio Application Object** and presents the script with access to this object via the **oDmApp** variable. The **oScript** variable gives access to the **Script Helper** object.

Adding the OK button onClick event

In this exercise we will add an **onClick** event that will be fired when the **OK** button is pressed. When the button is pressed, the script saves the point and/or string data, process it

using **CONPOL** and other commands, then re-load the resulting polygon back into the **Design Window**. This code is quite straightforward, but reasonably lengthy.

1. Using the left hand drop-down list and select the item [btnCreateShovel].
2. Using the right hand drop-down list select the item [onclick]. The following text has been added to the code in the main window.

```
function btnOK_onclick() {
}
```

3. Within the body of the **btnOK_onclick()** function (between the '{' and '}' characters), copy and paste the following text:

```
oDmApp.ActiveProject("delete-file '_tmp*'");
// Remove temporary work files
// alert("The current filter is " + tbFilter.value);
if (cbIncludePoints.checked) // Create file from point data with XP and YP only
{
oDmApp.ActiveProject.Design.SaveAllPointsAsDmFile("_tmp001", true, "");
if (tbFilter.value != "")
{
oDmApp.ParseCommand_
("picrec &IN=_tmp001 &OUT=_tmp001a APPEND=0 '" + tbFilter.value + "' 'END'");
oDmApp.ParseCommand("copy &IN=_tmp001a &OUT=_tmp001");
}
oDmApp.ParseCommand_
("extra &in=_tmp001 &out=_tmp002 'XP=XPT YP=YPT erase(XPT,YPT,ZPT,SYMBOL) go'");
}
if (cbIncludeStrings.checked) // Create file from string data with XP and YP only
{
oDmApp.ActiveProject.Design.SaveAllStringsAsDmFile("_tmp003", true, "");
if (tbFilter.value != "")
{
oDmApp.ParseCommand_
("picrec &IN=_tmp003 &OUT=_tmp003a @APPEND=0 '" + tbFilter.value + "' 'END'");
oDmApp.ParseCommand("copy &IN=_tmp003a &OUT=_tmp003");
}
oDmApp.ParseCommand("extra &in=_tmp003 &out=_tmp004 {' + _
tbFilter.value + "'} 'erase(ZP,PTN,PVALUE,SYMBOL,LSTYLE) go'");
}
if (cbIncludePoints.checked && cbIncludeStrings.checked)
// Append string coordinates to point coordinates
{
oDmApp.ParseCommand("append &IN1=_tmp002 &IN2=_tmp004 &OUT=_tmp005
@SEQUENCE=0_ @PROTODD=0 @PRINT=0");
oDmApp.ParseCommand("copy &IN=_tmp005 &OUT=_tmp002");
}
else if (cbIncludeStrings.checked && !cbIncludePoints.checked)
{
oDmApp.ParseCommand("copy &IN=_tmp004 &OUT=_tmp002");
}
// All X,Y coordinates are now in _tmp002. Now run CONPOL.
var command = "conpol &IN=_tmp002 &PERIMOUT=_tmp001 *X=XP *Y=YP";
if (tbExtension.value != "")
command += " @EXTDIS=" + tbExtension.value;
oDmApp.ParseCommand(command);
oDmApp.ParseCommand("proper &PERIMIN=_tmp001 &PERIMOUT=_tmp002 @TOL=0.000001_
@CLOSE=1");
oDmApp.ActiveProject.Data.LoadFromProject("_tmp002");
// load new polygon into design window
oDmApp.ActiveProject("delete-file '_tmp*'");// Remove temporary work files
```

After this code has been added to the HTML document click **File | Save** from the menu. The data will then be changed within FrontPage 2003.

Select **File | Save** once more from within FrontPage 2003 to save your data.

Operation of the Script

The **btnOK_onclick()** event handler script creates and uses several temporary files, all called "_tmp00n", which are deleted both at the beginning and at the end of the script.

If the *Include point data* check box is checked, the script writes all point data to a file, using the **PICREC** command the file is then filtered as specified in the "Filter" text box, if any. This file is then processed by **EXTRA** to remove all but the *XP* and *YP* fields.



If the *Include string data* check-box is checked, the script writes strings to another file, again using the command **PICREC** and whatever filter has been specified, and uses **EXTRA** to remove and rename fields. If both check-boxes were ticked, it appends all the data into one file.

Next, the script runs **CONPOL**. Notice how the basic command is assigned to a JavaScript variable, and the value for the **EXTDIS** parameter is added if appropriate.

CONPOL does not close the resulting polygon. Hence the script uses **PROPER** to close it before re-loading the polygon into the **Design** Window.

Finally, the script deletes all the temporary files created during the execution.

Running the Polygon Script

You can run the script in one of two ways:

- first, you can go to Studio 3 and select **Tools | Scripting | Run Script...** and specify the name of the HTML file that you have created. This is how it would normally be run, and this is the only way that you can check that the HTML interface looks right inside Studio 3.
- As an alternative, for a quick view, there is another way to run the script; by selecting the *Preview* tab in FrontPage 2003®. This is a convenient way to check that everything works the way that you expect, without having to remember to save the file, switch to Studio 3, and reload the file each time you make a change.



If you are running your script from the *Preview* tab of FrontPage 2003, remember that you must also have Studio 3 running. Also, if you are editing and saving the script in FrontPage 2003 and running the script in Studio 3 then you must refresh the script in Studio 3 every time it is saved.

Debugging the Script with Alert

Microsoft provides a number of sophisticated debugging environments, which are by far the best way of understanding where a script is going wrong. But for this example, the simplest debugging aid is the JavaScript **alert()** function that we used earlier. Using **alert()** you can display messages and the values of variables in a pop-up message box which is created at

some point during the execution of your script. For example, to check the value of the Filter text box before attempting to use it, you can add the following statement usually within or close to the place that it is used such as the **btnOK_onclick** event handler:

```
alert("The current filter is " + tbFilter.value);
```

When executed, this statement produces a message box similar to the following:



Debugging the Script with Microsoft Script Editor (MSE)

A full discussion of how to use the Script Editor for debugging is outside the scope of this tutorial. For full details, refer to the on-line Help. However, to give a flavour of what is possible, let's investigate some of the features available.

1. First, activate the Script Editor from FrontPage 2003, by selecting **Tools | Macro | Microsoft Script Editor....**
2. To start debugging, select **Debug | Start** in the Script Editor. This causes the Script Editor to reconfigure itself, and start an Internet Explorer window containing your HTML and script.
3. You can return to the Script Editor at any time, and insert a breakpoint on a particular line. To insert a breakpoint, click in the grey left-hand margin adjacent to the line where you want execution to be interrupted. For example, to break at the first line of the **btnOK_onclick()** event handler, the Script Editor view should look like the following. The red circle indicates a break point:

```
function btnOK_onclick() {  
    oDmApp.ActiveProject.DeleteFile("_tmp*");  
    alert("The current filter is " + tbFilter.value);  
    |  
    if (cbIncludePoints.checked) // Create file from  
    {
```

4. When script execution reaches the breakpoint, execution will pause and control is returned to the Script Editor. You can then look at the values of variables, single-step through the script, even change the flow of execution and the values of variables. Full details are given in the Script Editor on-line help.
5. To finish debugging, select **Debug | Detach All** from the Script Editor menu.

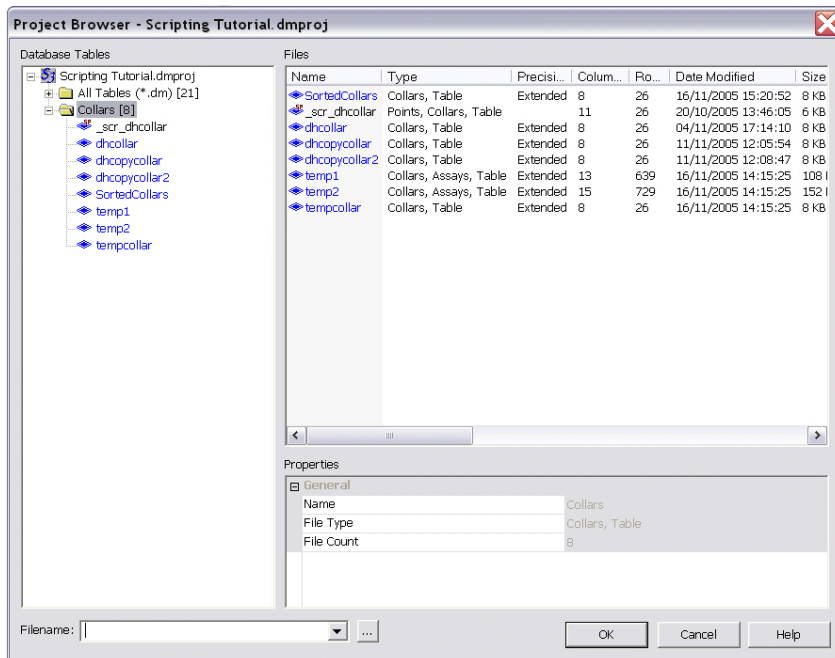
Suggestions for Further Extensions

This worked example is designed to be an instructive lesson. The resulting script has a number of limitations. You might like to try some of the following improvements by yourself:

- Use separate filters for points and strings.
- Allow the user to specify a value for the @MAXLEN parameter in **CONPOL**.
- Enhance the script to work in any orthogonal plane (hint: use radio buttons).
- Enhance the script to work in any plane (hard).

7 EXAMPLE: BROWSING AND COPYING

In this section, you will use the **Project File Browser** to browse for files.



The **Project File Browser** is a graphical interface into Studio 3 projects, and can be accessed using a variety of Studio 3 working methods, including (but not limited to):

- Selecting **Data | Load**, followed by a proprietary Studio 3 object type (wireframes, points, strings etc.)
- right-clicking the **Design** window data area and selecting the **Load** menu option, followed by one of the listed sub-items
- selecting **File | Browse Project Files...**
- right-clicking the top-level project icon in the **Sheets** or **Loaded Data** control bars and selecting **Load | From Project**.

ActiveX Controls

The browser you are going to display and manipulate is an *ActiveX* control. At this point, most guides describe the advent of the **ActiveX** control as a 'standalone, reusable component, using an accepted and constant data model'. Sadly, however, it's just not possible to give a clear technical definition of what the term **ActiveX** means. The reason for this is simple: **ActiveX** is a marketing label, not a technical term.

The clearest way to think about it is to view **ActiveX** as a brand name, like, say 'Windows'. Microsoft assigns the Windows trademark to a varied collection of operating system versions, and what that collection includes changes over time. Still, all Windows packages have some common elements, and assigning them a common name makes Microsoft's marketing task easier.

But, this is getting too far off the track, so for now, we can think of an **ActiveX** control in a more familiar (although slightly less accurate sense) – a prefabricated bundle of code, that

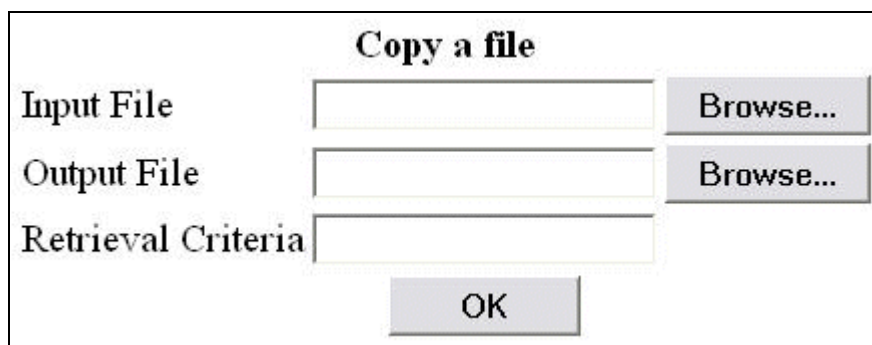
can be accessed through common scripting methods, including Javascript. The important point to remember about these controls is that they can not only form part of a script run within the confines of a Studio 3 application; they can also be accessed without the need for a parent application. In other words, you can get to the functionality provided using a simple scripting interface such as an HTML file.

This exercises that follow allow you to create a script that will copy a file with optional retrieval criteria. It uses the **Project File Browser** control using something you have already accessed, and included in your scripts - the **Studio 3 Application Object**. In true scripting terms, we would speak of the **Studio 3 Application Object** providing a 'handle' to the **Project File Browser** control – all this means is that one of the methods available to the Studio 3 Application Object allows you to easily launch the browser control, according to some settings you will specify in your script (more scripting speak: The Project Browser launching method accepts a series of user-definable arguments).

The example that follows can easily be extended to carry out any file processing along the way, or, for example, to create a script that reads multiple input files and creates one or more output files. It is worthwhile going through the exercises listed, then trying out some scripts of your own. In the future, if any of your scripts require a user to select a file, upon which to run a script, or make the subject of a Studio 3 process, the **Project Browser object** is a very useful tool.

In the following exercises, you are going to:

- Create a **Browse and Copy** Interface – this will allow users of your script to specify both an input and an output file using the **Project Browser**. The interface you will be creating will look similar to the image shown below:



Copy a file	
Input File	<input type="text"/> <input type="button" value="Browse..."/>
Output File	<input type="text"/> <input type="button" value="Browse..."/>
Retrieval Criteria	<input type="text"/>
<input type="button" value="OK"/>	

- Run and test the script.
- Explore some possible variations of the script.

Creating the Browse and Copy Interface

This section takes you through creating an interface similar to that shown in the preceding section. This interface includes some of the common HTML interface controls; a table, some buttons, an editable text field and some text labels.

Before you start

The exercises referred to in this section assume the following tasks have been undertaken already:

- The file `_vb_scr_points.dm` will be used to test the script. This file can be found at `C:\Database\DMTutorials\Data\VBOP\Datamine` for standard installations.

This section includes the following exercises:

- Drawing a table to contain the interface objects
- Adding text and text boxes
- Adding browse buttons
- Finishing and tidying up your interface
- Assigning names and values to the controls

Drawing a table to contain the interface objects

You will use the *Design* tab to insert a table into the interface.

1. Start Microsoft FrontPage 2003.
2. Create a new blank page using **File | New...** from the menu and selecting *Blank Page* from the **New Page** window.
3. Save the file using **File | Save As...** from the menu and in the **Save As** dialog enter the name 'Browse.htm' and locate the file in the `C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts` directory.
4. Click **Save**. The file extension should be defined for you. Note that a partial HTML sections have already been added to the file.
5. This file will be used in subsequent tutorial exercises.
6. Select **Table | Insert | Table...** from the menu. In the **Insert Table** dialog increase the value of *Rows* to 5 and *Columns* to 3 click **OK**. This will have inserted a table containing five rows and three columns into the *Design* tab.

Adding text and text boxes

Next we will add some text and text boxes to the table. The text describing the contents of the text boxes will be placed into the left hand cells of the table and the text boxes themselves into the right hand cells.

1. Position the cursor within the top left hand cell of the table and type the words 'Copy a File'. You can highlight the text and select **Bold** from the toolbar, if you wish.
2. We now want to merge all the cells in the top row. Use the mouse to highlight the top row of cells, then right-click and select **Merge Cells** from the popup menu. With the cursor in the top row, centre the text using **Format | Paragraph...** from the menu, and selecting an *Alignment* of *Center*. Click **OK**.
3. Position the cursor in the left hand cell in the second row and type 'Input File'.
4. Position the cursor in the left hand cell in the third row and type 'Output File'.
5. Position the cursor in the left hand cell in the fourth row and type 'Retrieval Criteria'.

6. Position the cursor in the middle column of the second row and select **Insert | Form | Text Box** from the menu. This will place a text box within the cell. This will be used to display the name of the input file.
7. Repeat step 6 for the middle column, third row. This will be used to display the name of the output file.
8. Repeat step 6 for the middle column, fourth row. This is where we will enter retrieval criteria.

Adding browse buttons

Next, you will need to add some buttons for activating the Project Browser control:

1. In the right hand cell, second row, select **Insert | Form | Push Button** from the menu. Do not worry about the size and position of the button for now. We will adjust the names of these buttons in a later step.
2. Repeat step 1 by selecting the right hand cell, third row.
3. Repeat step 1 by selecting the middle column and fifth (last) row.

Finishing and tidying up the interface

Now, we can improve the look of the table. This step is not absolutely necessary, but it will improve the visual integration with Studio 3, and will make the interface behave better if the script window in Studio 3 is resized.

All of the table layout improvements can be done with right-clicks of the mouse, and changing values in the resulting dialogs. However, this lesson will give the menu command alternatives.

1. Select all the cells in the table by positioning the cursor in the top left cell and with the left mouse button depressed drag and select all the cells in the table.
2. Right-click in the table and select **Table Properties** from the menu. In the **Table Properties** dialog, clear the *Specify width* and *Specify height* check boxes to allow the browser to select the best fit width and height for the cells. With this example we shall also remove the borders around the table. In the *Borders* section reduce the *Size* to 0. Click **OK**. You will notice that in the design window a dashed outline is present to signify the table borders, this will not be present when the table is displayed in the *Preview* tab.
3. Position the cursor in the middle column, fifth row (containing the button) and using **Format | Paragraph...** from the menu, select an Alignment of *Center*. Click **OK**
4. To give the HTML page a name; right-click the *Design* tab, and select **Page Properties** from the popup menu. In the **Page Properties** dialog enter the text 'Browse and Copy' into the *Title* field. Click **OK**.
5. Save the interface by selecting **File | Save** from the menu.
6. Click on the *Preview* tab to see how the interface will look.

Assigning names and values to the controls

You now need to assign names and initial values to all of the interface controls and change the button labels. Names can be anything that you wish, but remember that upper and lower case letters are distinct (the name 'this' is different from 'This'). A good idea is to prefix the names of controls with two or three letters that identify the type of control, e.g. **btnOK** for an **OK** button, or **tbSelectedFile** for a text box that will contain the name of a selected file.

There are a few ways to assign a name to a control we shall use either the right-click method on the control and use the **Form Field Properties...** from the menu. Alternatively you could just double-click on the control with the left mouse. Either of these methods will display the relevant properties dialog.

1. Double-click the *Input File* text box in the middle column in the *Design* tab. In the **Text Box Properties** dialog type 'tbInputFile' into the *Name* field. Click **OK** to accept the changes.
2. Double-click the *Output File* text box in the middle column in the *Design* tab. In the **Text Box Properties** dialog type 'tbOutputFile' into the *Name* field. Click **OK** to accept the changes.
3. Double-click the *Retrieval Criteria* text box in the middle column in the *Design* tab. In the **Text Box Properties** dialog type 'tbRetrieval' into the *Name* field. Click **OK** to accept the changes.
4. Double-click the button in the top right of the table. In the *Name* field type 'btnBrowse1'. In the *Value/label* field type 'Browse...'.
5. Double click the button in row three. In the *Name* field type 'btnBrowse2'. In the *Value/label* field type 'Browse...'.
6. Finally, double-click the button at the bottom and in the *Name* field type 'btnOK'. In the *Value/label* field type ' OK ' (add some spaces either side of the text to space the button out).
7. Save the interface by selecting **File | Save** from the menu.
8. Click the *Preview* tab to see how the interface will actually look. It should now look like the figure at the top of this page.
9. Click the *Code* tab and look at the values that have been entered into the HTML code for you by designing the interface.

Adding the Browse and Copy Script

Make sure that you have saved the completed script. If you want to use the reference version then take a copy of '_scr_Browse_and_Copy 0001.htm' into your Tutorial database.

Before you start

The exercises referred to in this section assume the following tasks have been undertaken already:

- The file *_vb_scr_points.dm* will be used to test the script. This file can be found at **C:\Database\DMTutorials\Data\VBOP\Datamine**
- The *Browse.htm* file created during the exercise *Creating the Browse and Copy Interface*.

Running and testing the script

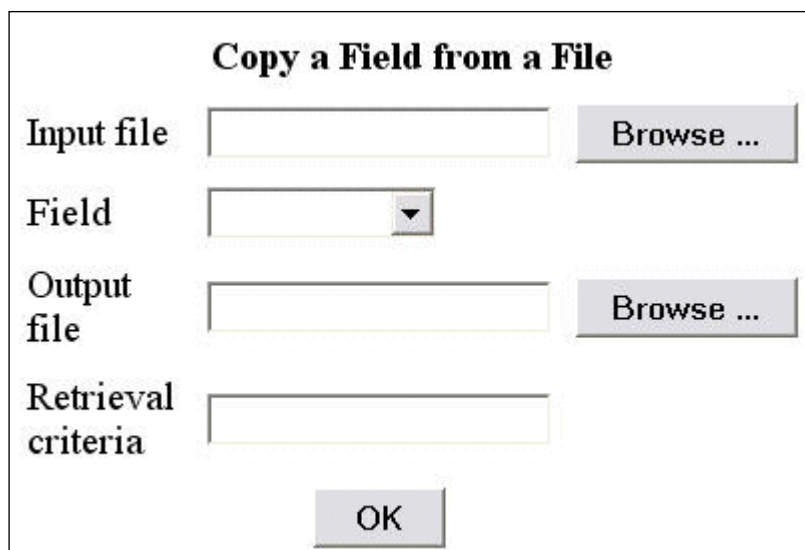
We will perform three tests on the script to ensure it is working as expected.

1. From Studio 3, select **Tools | Scripting | Run Script...** from the menu.
2. Select the *Browse.htm* file created in *Creating the Browse and Copy Interface* and click **Open**. If the **Script Analyser** dialog appears, click **No**.
3. Click **Browse...**, from the browser dialog select the file *_vb_scr_points* from **C:\Database\DMTutorials\Data\VBOP\Datamine** and click **OK**.
4. We can either specify the name of an *Output File* or use the second **Browse...** button. For this example we will type the name of the file as 'OutFile'.
5. Leave the *Retrieval Criteria* text box blank.
6. Before executing the script, ensure that the **Command** window is visible and right-click in the window and select **Clear** from the popup menu.
7. Click the **OK** button in the script window. The **Command** window should now display the information that 80 records were copied to the file *outfile.dm*.
8. Leaving the *Input* and *Output file* details as they are, type 'COLOUR=19' into the *Retrieval Criteria* text box.
9. Click the **OK** button in the script window. The **Command** window should now display the information that 38 records were copied to the file *outfile.dm* and the currently established retrieval criteria was '19'.
10. Repeat step 8 and 9 using 'COLOUR=11'. The **Command** window should now display the information that 17 records were copied to the file *outfile.dm* and the currently established retrieval criteria was '11'.

Further Browse and Copy Extensions

The aim of this Chapter has been to familiarize you with interface design and the setting up of event handlers. However the resulting script has a number of limitations. You might like to try the following improvements by yourself.

Copying a single field



Copy a Field from a File

Input file **Browse ...**

Field

Output file **Browse ...**

Retrieval criteria

OK

This extension will copy a single field from a file. The script changes the copy command into the selective copy command or **SELCOP**.

First the interface needs to be extended by inserting an additional row in the table between the *Input* and *Output files*. In the left hand cell of the new row the word 'Field' should be typed and then use **Insert | Form |**

Drop-Down Box to define a selection box in the middle column. This box should be given the name 'selFields'. The interface should look like the one shown on the left (although you may need to set the width of the drop-down box to 115 pixels to give it some initial width).

In order to fill in the list with the names of the fields from the input file, use the following function call in the **btnBrowse1_onclick()** function:

```
oScript.makeFieldsPicklist(tbInputFile.value, selFields);
```

The entire event handler code should now look like this:

```
function btnBrowse1_onclick() {  
    tbInputFile.value = DisplayBrowser();  
    oScript.makeFieldsPicklist(tbInputFile.value, selFields);  
}
```

The **makeFieldsPickList** has two arguments; the first is the input filename and the second is the name of the select (drop-down box) list where the names are to be added. It is possible to add a third, optional argument to this method, which gives the default value to be displayed. The default value need not necessarily be the name of a field in the file.

Finally, we need to modify the **btnOK_onclick()** handler for the **OK** button. The 'copy' command becomes a 'selcop' command, and we need to add the name of the selected field to the command. The new code looks like this:

```
var theCommand = "selcop &in=" + tbInputFile.value + " &out=" +  
tbOutputFile.value +  
  
" *f1=" + selFields.value + " @keepall=1";
```

The parameter **keepall** (set to 1) instructs the command to keep all data and not discard duplicates in the output file. The complete **btnOK_onclick()** handler should look like the following:

```
function btnOK_onclick() {  
    if (tbInputFile.value == "" || tbOutputFile.value == "") {  
        alert("Missing file name");  
        return;  
    }  
    var theCommand = "selcop &in=" + tbInputFile.value +  
" &out=" + tbOutputFile.value +  
" *f1=" + selFields.value + " @keepall=1";  
    if (tbRetrieval.value != "")  
        theCommand += " {" + tbRetrieval.value + "}";  
    oDmApp.ParseCommand(theCommand);  
}
```

Note that we have also added an **alert** method to inform the user when input or output files are missing:



The reference file for this script is *_scr_Browse_and_Copy 0002.htm*, found in your project directory.

Copying multiple fields

For the following extension, the changes to the user interface are easy; in the **Properties** dialog for the drop-down box element, set the *Height* to 4 lines, and select the *Allow multiple selections* radio button. The User Interface now looks like this:

The screenshot shows a dialog box with the title "Copy Fields from a File". It contains the following elements:

- Input file:** A text input field followed by a "Browse ..." button.
- Fields:** A multi-line drop-down menu.
- Output file:** A text input field followed by a "Browse ..." button.
- Retrieval criteria:** A single-line text input field.
- OK:** A button at the bottom center of the dialog.

No changes are needed in the script, apart from adding code to extract the values from the selected items in the drop-down list object. This is done by making further changes to the **btnOK_onclick()** handler. The code should look like the following:

```
function btnOK_onclick() {
    if (tbInputFile.value == "" || tbOutputFile.value == "") {
        alert("Missing file name");
        return;
    }

    var theCommand = "selcop &in=" + tbInputFile.value + " &out=" +
    tbOutputFile.value;

    // Extract selected options from multiple-select menu

    var opts = selFields.options.all;
    // "opts" is the collection of all options

    var fieldNum = 0;

    for (var i = 0; i < opts.length; i++)
    // opts.length is the total number of options

        if (opts(i).selected)

            theCommand += " *f" + (++fieldNum) + "=" + opts(i).value

    if (fieldNum == 0) {
```

```

    alert("No fields selected.");

    return;
}

if (tbRetrieval.value != "")

    theCommand += " {" + tbRetrieval.value + "}";

oDmApp.ParseCommand(theCommand);
}

```

The field *opts* is first set to the collection of all options within the *selFields* drop-down list. This list box has a property called *selected* which informs us as to which items are actually selected by the user. Those that are selected are appended to the command for retrieval by the **SELCOP** command. Note, that if no fields are selected the routine returns without doing anything.



The reference file for this script is *_scr_Browse_and_Copy 0003.htm*, found in the **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts** directory.

8 EXAMPLE: RUNNING A MACRO

You may already have macros that you use for running sequences of commands. Input to these macros is usually through either the prompt or screen command. However as you will have already realized the HTML interface provides a far more attractive, comprehensive and easy to use method of entering files, fields, parameters and other runtime variables.

In order to take advantage of the HTML interface you don't have to convert your entire macro to script. You can simply set up the interface in HTML and pass the values of the variables to your macro. Therefore the only changes you need to make to your macro are to remove the prompt or screen commands and allow the macro to read the variable values which you have entered through the HTML interface. At the end of the macro you can pass the values of variables back to the script.

Scripting uses the 'var' file to transfer variables and their values between the script and the macro. This is the file already used by the **varload** and **varsave** macro commands for saving information between Sudio sessions. It is also the file used by the older versions of these commands, **STKPAR** and **STKSAV**. The **varload()** and **varsave()** methods provide the equivalent functions in scripting, and are included as part of the **Application Object**. The macro will be executed using the **ParseCommand** method of the **Application Object**.

Using XRUN to Execute the Macro

We will first examine the Macro that we are going to use. This is located in your project directory and called *_scr_Shovel.MAC*. This file in turn uses a wireframe definition, and can be found under C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts.

The macro uses the standard techniques for setting default values and prompting. The main steps are:

- Define default values for substitution variables.
- Test whether the 'var' file exists, and if it does replace the initial defaults with the saved values.
- Use prompt to define current values, with defaults as defined in the previous steps.
- Create and display the wireframe model at the required location.
- Save the substitution variables.

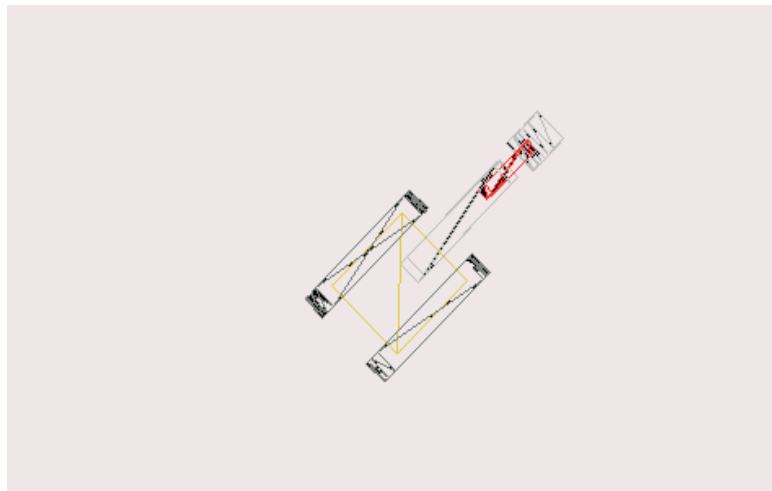
You will see that the five substitution variables that are prompted for using the prompt command are:

- **\$APPEND#** - decision whether to append shovels into a single file or overwrite the file. A zero overwrites the file.
- **\$E#** - position of the shovel in Easting.
- **\$N#** - position of the shovel in Northing.
- **\$Z#** - position of the shovel in Elevation.
- **\$AZIM#** - the orientation of the shovel in degrees. Zero degrees represents North.

In the following exercise we will remove these prompts from the macro and create a script with an HTML interface that will run the macro.

To see how the macro works, save a copy of the macro file `_scr_Shovel.MAC` as `Shovel.mac` in your Tutorial project directory.

Then, run the macro by typing 'XRUN' into the **Run Command** window and taking all the default values to the prompts. The macro will create a wireframe file called `shovtr.dm` and `shovpt.dm`. At the end of the macro this file will not be displayed automatically. To display the



file use **File | Add to Project | Existing Files...**, select the files 'shovtr' and 'shovpt' and add them to the project. Then drag one of the files onto the **Design** window. You should now see a screen image similar to the one shown on the right.

The text in the **Output** window shows the progress of the macro. The figure above represents a shovel located at -50 Easting, 20 Northing and 0 Elevation with an Azimuth of 45 degrees.

Creating the macro interface

The interface you are going to create is shown below:

Creating a Wireframe Shovel	
Shovel Azimuth	<input type="text" value="33"/>
Shovel Elevation	<input type="text" value="0"/>
	<input checked="" type="checkbox"/> Append Shovels
<input type="button" value="Create Shovel"/>	
<input type="text"/>	

This interface contains three of the five substitution parameters that we are going to use in the macro. These are the azimuth and elevation of the shovel. The Easting and Northing data will actually be obtained from the **Design** window. The *Append Shovels* field will be used to determine whether data overwrites or is appended to the file.

This section takes you through the following exercises:

- Drawing a table to contain the interface objects
- Adding some text and text boxes
- Adding a check box

- Adding a push button
- Finishing and tidying up the interface
- Assigning names and values to the controls

Before you start

- The macro file *Shovel.MAC* will be used in this exercise. This file can be found under **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts**. Copy the file *_scr_Shovel.MAC* to *Shovel.MAC* now so that we can change the body of the macro when required.
- The Datamine file *shovtr.dm* and *shovpt.dm* will contain the output wireframe data. These files can be found under **C:\Database\DMTutorials\Data\VBOP\Datamine**.

Drawing a table to contain the interface objects

You will use the *Design* tab to insert a table into the interface:

1. Start Microsoft FrontPage 2003.
2. Create a new blank page using the **File | New...** from the menu and selecting *Blank Page* from the **New Page** window.
3. Save the file using **File | Save As...** from the menu and in the Save As dialog enter the name 'Shovel.htm' and locate the file in the **...Projects\S3ScriptTut\ProjFiles\MyProj1** directory. Click the **Save** button. The file extension should be defined for you. Note that a partial HTML sections have already been added to the file.
4. Select **Table | Insert | Table...** from the menu. In the **Insert Table** dialog increase the value of *Rows* to '6' and click **OK**. This will have inserted a table containing six rows and two columns into the *Design* tab.

Adding some text and text boxes

Next, you will add some text and text boxes to the table. The text describing the contents of the text boxes will be placed into the left hand cells of the table and the text boxes themselves into the right hand cells where appropriate.

1. Position the cursor within the top left hand cell of the table and type the words 'Creating a Wireframe Shovel'. This will eventually be the title.
2. In the next row down, type the words 'Shovel Azimuth'.
3. In the next row down, type the words 'Shovel Elevation'.
4. Position the cursor in the right hand cell of the second row and select **Insert | Form | Text Box** from the menu. This will place a text box within the cell.
5. Now repeat step 4 for row three and insert another text box in the cell below.
6. Repeat step 4 for the last row, left hand column and insert another text box.

Adding a check box

Next, you will add a check box to the table. The text describing these will be associated with the check box itself:

1. Position the cursor in row four, right hand column, directly below the text box.
2. Select **Insert | Form | Checkbox** from the menu. With the cursor in the cell, type in the text 'Append Shovels' to its right.

Adding a push button

Next, you will add a push button for activating the form:

1. In the fifth row, left column, select **Insert | Form | Push Button** from the menu. Don't worry about the size and position of the button for now, as you will adjust the names of these buttons in a later step.
2. Your interface should now look very similar to the one shown at the start of this section but without some of the formatting.

Finishing and tidying up the interface

Now, we can improve the look of the table. This step is not absolutely necessary, but it will improve the visual integration with Studio 3, and will make the interface behave better if the script window in Studio 3 is resized.

All of the table layout improvements can be done with right-clicks of the mouse, and changing values in the resulting dialogs. However, this lesson will give the menu command alternatives.

1. Select all the cells in the table by positioning the cursor in the top left cell and with the left mouse button depressed drag and select all the cells in the table.
2. Select **Table| Table Properties | Cell** from the menu. In the **Cell Properties** dialog, clear the *Specify width* and *Specify height* check boxes to allow the browser to select the best fit width and height for the cells. Click **OK**.
3. Highlight the two columns in the top row, right click and select **Merge Cells** from the popup menu. Select the title text in the cell and select **Format | Font...** from the menu, use a Font style of 'Bold' and click **OK**.
4. Highlight the two columns in the fifth row, right click and select Merge Cells from the popup menu. Select the button text in the cell and select **Centre** from the **Format** toolbar.
5. Repeat step 4 for the bottom row. Select the text box and drag the right hand side to the extent of the merged cells.
6. To give the HTML page a name; right-click the *Design* tab, and select **Page Properties** from the popup menu. In the **Page Properties** dialog enter the text 'Shovel Script' into the *Title* field. Click **OK**.
7. Save the interface by selecting **File | Save** from the menu.

Assigning names and values to the controls

Before we can make anything happen, we need to assign names and initial values to all of the interface controls. Names can be anything that you wish, but remember that upper and

lower case letters are distinct (the name 'this' is different from 'This'). A good idea is to prefix the names of controls with two or three letters that identify the type of control, e.g. **btnOK** for an **OK** button, or **tbSelectedFile** for a text box that will contain the name of a selected file.

There are a few ways to assign a name to a control we shall use either the right click method on the control and use the **Form Field Properties...** from the popup menu. Alternatively, you could just double-click the control with the left mouse. Either of these methods will display the relevant properties dialog.

1. Double click the *Shovel Azimuth* text box in the *Design* tab. In the **Text Box Properties** dialog type 'tbAzimuth' into the *Name* field and type '0' (zero value) into the *Initial value* field. This will initialise the text box with data when it is displayed. Click **OK** to accept the changes.
2. Double-click the *Shovel Elevation* text box in the *Design* tab. In the **Text Box Properties** dialog type 'tbElevation' into the *Name* field and type '0' (zero value) into the *Initial value* field. This will initialise the text box with data when it is displayed. Click **OK** to accept the changes.
3. Double-click the *Append Shovels* check box in the *Design* tab (you may have to click the actual box that represents the check box itself rather than the text). In the **Check Box Properties** dialog type 'cbAppend' into the *Name* field. Select the *Not Checked* option for the Initial state. Click **OK** to accept the changes.
4. Double-click the Button in the *Design* tab. In the **Push Button Properties** dialog type 'btnCreateShovel' into the *Name* field and type ' Create Shovel' in the *Value/label* field. Click **OK** to accept the changes.
5. Double-click the text box in the bottom row in the *Design* tab. In the **Text Box Properties** dialog type 'tbInfo' into the *Name* field and set the *Width in characters* field to 40. Click **OK** to accept the changes.
6. Save the interface by selecting **File | Save** from the menu.
7. Click the *Preview* tab to see how the interface will actually look. This should now be the same as the figure at the top of this exercise.
8. Click the *Code* tab and look at the values that have been entered into the HTML code for you by designing the interface.

Adding the Macro Script

You now have an HTML user interface, together with all the hooks for the 'Run Macro' script.

The process for adding this script is very similar to that outlined in 'Adding the Polygon Script' in Chapter 7, but it is useful to recap some of the fundamental points behind the process of adding script event handlers.

Script events recap

The script language used is not a part of Studio 3 - you can use any of a number of standard programming languages: **JavaScript**, **VBScript**, **PerlScript**, PowerBuilder's **PowerScript**, etc. The two languages supported by FrontPage 2003 are VBScript and JavaScript.

Unlike macros and menus, scripts are usually made up of a number of *event handlers*. These are small sections of script that are executed by the browser when some relevant event occurs.

Only two events are handled in this script:

- When the window loads (a window **onload** event occurs). We use this event to make a connection to the **Application Object**.
- When the **OK** button is pressed (a button **onClick** event occurs). We use this event to perform the tasks. These include picking a point in the **Design** window to locate the shovel, saving the macro 'var' file, calling the macro and finally loading the wireframe file back into the **Design** window.

Before you start

The exercises referred to in this section assume the following tasks have been undertaken already:

- Ensure Microsoft Script Editor is installed and integrated with Microsoft FrontPage 2003®. In FrontPage 2003® you should find the menu item under **Tools | Macro | Microsoft Script Editor**. The menu looks similar to the one shown below:



- We will continue to use the *Polygon.htm* file created in the previous exercise.

The following exercise is repeated from the section entitled 'Adding the Polygon Script' in Chapter 7. If you have already set up and configured Microsoft Script Editor on your system, you can skip this stage and move onto 'Adding the windows **onload** event'.

Setting up MSE

To start entering the script that makes the HTML come alive, first click on the *Code* tab, then select **Tools | Macro | Microsoft Script Editor** from the menu. A new window appears, showing your interface in HTML. We will use this application to define code sections for button click and window events.

5. To configure the Script Editor to use JavaScript, select **View | Property Pages** from the menu.
6. In the **DOCUMENT Property Pages** dialog that is displayed, on the *General* tab, set the *Client* field to 'JavaScript'. Click **OK**. You will need to perform this task every time the Script Editor is used on a different document. The Script Editor does not save JavaScript as the default.
7. Select **View | Toolbox** from the menu. This will display a list of objects such as buttons and check boxes etc.
8. Select **View | Other Windows | Document Outline** from the menu. This will display the list of objects that have been added to the interface.

Adding the windows onload event

In this exercise we will add an event that will be fired when the window is loaded.

8. Above the main window (the one displaying the code), you will notice two drop down list boxes. The left hand one contains the text 'Client Objects & Events' and the right hand one contains the text '(No Events)'.
9. Using the left hand drop-down list and select the item [window].
10. Using the right hand drop down box select the item [onload]. The following text has been added to the code in the main window. If after selecting the [onload] item, the words 'Sub window_onload()' appears in the code window, Script Editor is configured to use VBScript.

```
function window_onload() {
}
```

11. We now need to enter the JavaScript code to be executed when this event occurs. First, we need to establish a connection between the script and the **Studio Application Object**. To accomplish this we must define a CAE Studio application variable, and refer to a routine to instantiate the object.
12. Before the function **window_onload()** type the following text

```
var oDmApp = null;
var oScript = null;
```

13. Within the body of the **window_onload()** function (between the '{' and '}' characters), type the text 'AutoConnect();'. This instructs the **onload** event to call a function **AutoConnect** that we will now write. This routine will use the variable **oDmApp** defined in step 5.
14. After the **window_onload()** function and after the '}' symbol copy and paste the following text:

```
function AutoConnect() {
  oScript = new ActiveXObject("DatamineStudio.ScriptHelper");
  oScript.initialize(window);
  oDmApp = oScript.getApplication();
  if (oDmApp== null)
    return false;
  else
    return true;
}
```

15. The final code should look like the following:

```
var oDmApp = null;
var oScript = null;
function window_onload() {
  AutoConnect();
}function AutoConnect() {
  oScript = new ActiveXObject("DatamineStudio.ScriptHelper");
  oScript.initialize(window);
  oDmApp = oScript.getApplication();
  if (oDmApp== null)
    return false;
  else
    return true;
}
```



The variables **oDmApp**, **oScript** and the **AutoConnect** function are necessary for every scripted solution that connects to the **Studio Application Object**. The **AutoConnect** function initialises the **Studio Application Object** and presents the script with access to this object via the **oDmApp** variable. The **oScript** variable gives access to the **Script Helper** object.

Adding the Create Shovel button onClick Handler

In the following exercise, you will add an **onClick** event that will be fired when the **Create Shovel** button is pressed. When the button is pressed, the script executes a series of commands, some of which are interactive.

At the end of the script the data is loaded into the **Design** Window. This code is quite straightforward, but reasonably lengthy:

1. Using the left hand drop-down list and select the item 'btnOK'.
2. Using the right hand drop-down list select the item 'onclick'. The following text has been added to the code in the main window.

```
function btnCreateShovel_onclick() {  
}
```

3. Within the body of the **btnCreateShovel_onclick()** function (between the '{' and '}' characters), copy and paste the following text.

```
var intAppend = 0;  
if (cbAppend.checked) {  
intAppend = 1 ;  
}  
if (tbAzimuth.value == "") {  
alert ("Enter a valid Azimuth for the shovel") ;  
return ;  
}  
if (tbElevation.value == "") {  
alert ("Enter a valid Elevation for the shovel") ;  
return ;  
}  
// provide prompts  
tbInfo.value = "Please select a point in the design window" ;  
objVars = new Object() ;  
objVars.APPEND = intAppend ;  
objVars.AZIM = tbAzimuth.value ;  
// Get the point in the design window that  
//will represent E and N we will override Z  
strPoint = oDmApp.ActiveProject.Design.Selection.PickPoint("Select a  
point");  
var aryTemp = strPoint.substr(7).split(",");  
objVars.E = parseFloat(aryTemp[0]);  
objVars.N = parseFloat(aryTemp[1]);  
// objVars.Z = parseFloat(aryTemp[2]);  
objVars.Z = tbElevation.value ;  
oScript.varsave(varfile, false, objVars);
```

```
// Now run the macro
oDmApp.ActiveProject.RunMacroFile("Shovel.mac", "T");
//now load the wireframe file into the design window and refresh the
//screen
tbInfo.value = "";
oDmApp.ExecuteCommand("unload-all-data");
oDmApp.ActiveProject.Data.LoadFromProject("shovtr");
oDmApp.ActiveProject.Data.LoadFromProject("shovpt");
oDmApp.ExecuteCommand("zoom-all-extents");
```

4. We also need to add a variable for the 'var' file and should be the name of the macro that we are running. We will also initialise the **tbInfo** text box so that no text is shown when the script loads as shown below.
5.

```
var varfile = "Shovel.var";
var oDmApp= null;
var oScript = null;
function window_onload() {
AutoConnect();
tbInfo.value = "";
}
```
6. After this code has been added to the HTML document click **File | Save** from the menu. The data will then be changed within FrontPage 2003.
7. Select **File | Save** once more from within FrontPage 2003 to save your data.

Operation of the Script

The btnCreateShovel_onclick() event handler script creates a wireframe shovel in the design window at the location specified by the user.

First the script checks the state of the Append Shovel check box and sets a variable called intAppend so that the value can be passed to the macro.



The script then checks that the Shovel Azimuth and Shovel Elevation fields are not empty and only allows the script to proceed if they have values in them.

The next step is interactive and we provide a prompt to the user by filling in the tbInfo text box with the words 'Please select a point in the design window'. At this point the user should click somewhere in the design window. The Studio command PickPoint provides a string with these parameters in it. The string is then parsed to obtain the individual X, Y and Z values which are placed into an object called objVars as E and N (notice that the Z field is overwritten with the elevation from the text box).

There are many ways to pass variables to a macro. In the script above, you have created an object and passed the object to the **varsave** function. The object should contain the names of the variables of the substitution parameters within the macro. Once the **varsave** command is invoked the *shovel.var* file will contain the parameters as set by the user. The macro is then executed. This action will either create a new shovel wireframe or append the new shovel to an existing file.

Finally we reset the *tbInfo* text box the system unloads all the data and reloads the 'shovtr' and 'shovpt' Datamine files and then zooms the screen so that all the data fits neatly.

Running the Macro Script

Before the script can be run we need to remove the macro Prompt commands. This is to ensure that the macro can be run using the parameters supplied by the user interface (the values entered into the form fields) and not request the same parameters in the command line, as would normally be the case for interactive macros.

Removing Macro Prompts

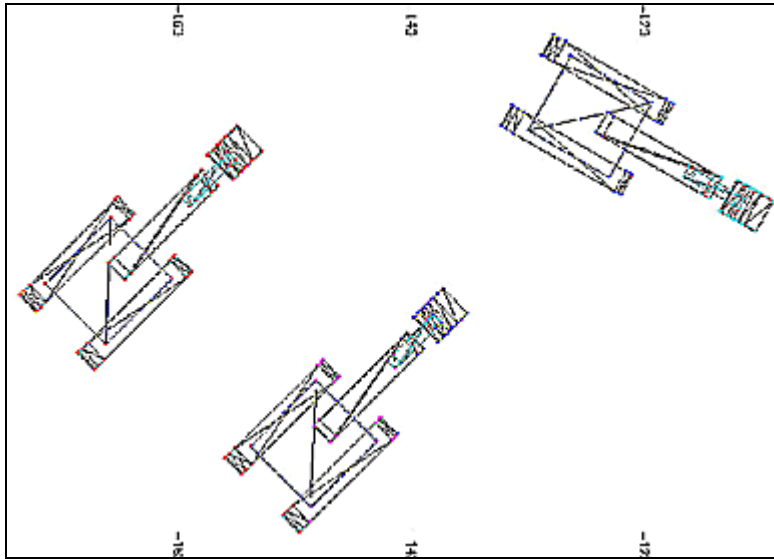
Before the script can be run we need to remove the macro Prompt commands. The following lines should be removed from the *Shovel.MAC* macro (found under C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts). This can be performed by editing the file in **Notepad**.

```
!PROMPT
0
1 Append Shovels (0=no, 1=yes) [$APPEND#] ? '$APPEND#',N
1 position of shovel in Easting [$E#] ? '$E#',N
1 position of shovel in Northing [$N#] ? '$N#',N
1 position of shovel in Elevation [$Z#] ? '$Z#',N
1 Azimuth that shovel is facing [$AZIM#] ? '$AZIM#',N
```

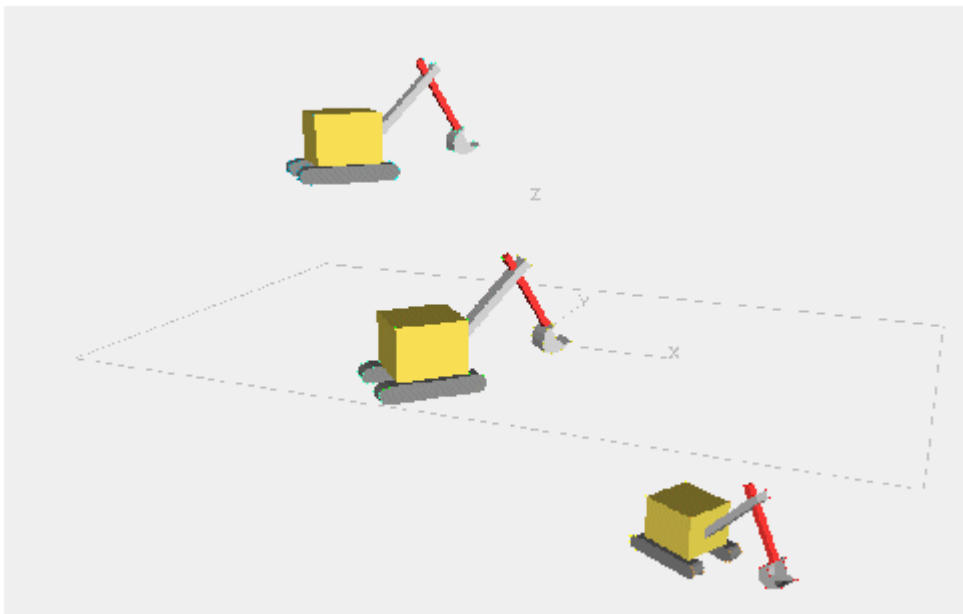
Running the Script

The script should be run by selecting Tools | **Scripting** | **Run Script...** from the menu. To test the script, follow the steps below.

1. Ensure the *Append Shovel* check box is cleared.
2. Enter a *Shovel Azimuth* of '45' degrees.
3. Click the **Create Shovel** button, and then click in the centre of the **Design** window.
4. Ensure the *Append Shovel* check box is checked.
5. Enter a *Shovel Elevation* of '20'.
6. Click the **Create Shovel** button, and then click in the top left hand corner of the **Design** window.
7. Notice that the **Design** window zooms to display both images appropriately.
8. Enter a *Shovel Elevation* of '-30'.
9. Enter a *Shovel Azimuth* of '120' degrees.
10. Click the **Create Shovel** button, and then click in the top right hand corner of the **Design** window. Three shovels have now been created that looks something like the following:



- Update the **Visualizer** by typing 'uv' while focused on the **Design** window. Adjust the view to see that shovels lie above and below the central plane as shown below:



Validation

We have not included any validation on the values that are entered into the script interface. Null or inappropriate values may cause problems when the macro is running. Validation can be done using JavaScript, but is not part of this exercise.

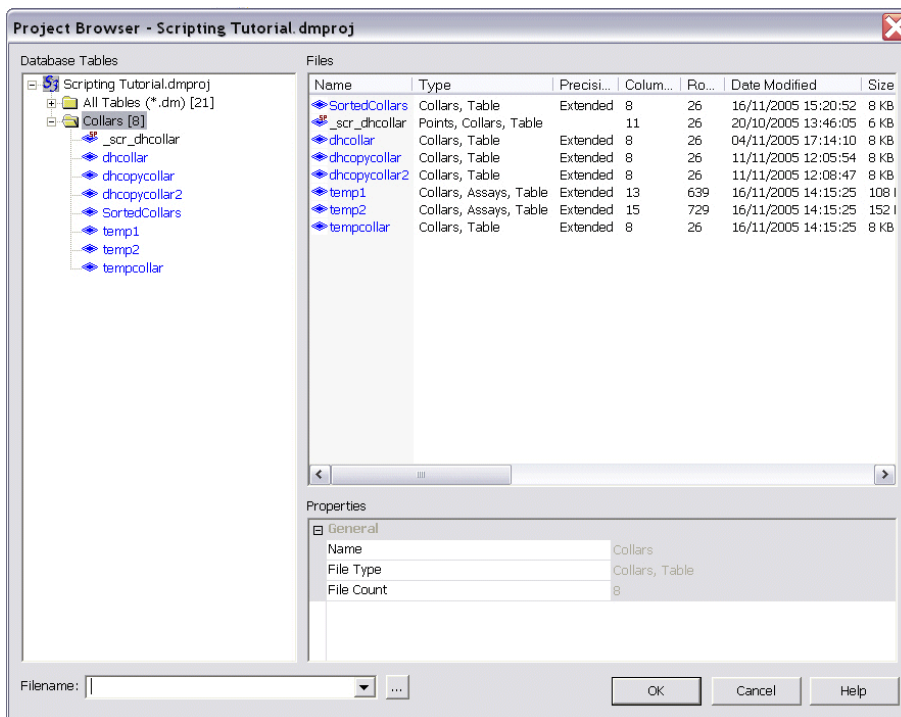
9 MORE EXAMPLES

Overview

One of the best ways of learning to write your own script is to look at examples. This section contains scripts which illustrate different features of the scripting language, and will cover:

- using the Studio 3 File Browser for selecting file names rather than the window `prompt()` method.
- providing an interface for defining the parameters of a model prototype, which then creates a wireframe model around the outside of the prototype model, and loads the wireframe into the **Visualizer**
- providing an interface for accessing the value of a field in a selected record from a Datamine file
- using the **ExecuteCommand** method vs. using the **ParseCommand** method.

Studio 3 File Browser Example



In the following example we will use the **Studio 3 File Browser** for selecting file names rather than the window `prompt()` method.

This is the same dialog that you would use if running commands interactively.

The use of the file browser has already been explored in Chapter 8 – Browsing and Copying. This section intends to expand on these

ideas and provide more in-depth information about how the browser control can be manipulated.

The **Studio 3 File Browser** is accessed using the **ActiveProject.Browser** Object of the **Studio Application Object**. The specific parts of the object we will be accessing are a **TypeFilter** and **FileName** property and the **Show()** method.

There are a number of tasks that have to be added to the starting file, such as:

- Creation of the new Browser Object.
- Setting properties and calling methods from within the Browser Object.
- Using the output of the Browser Object.

The following exercise demonstrates one way of using the Browser object.

Adding the Project File Browser

1. Start Microsoft FrontPage 2003.
2. Use the **File | Open...** menu command and using the **Open File** dialog browse to the file `_scr_SortCollarsPromptAlertConfirm.htm` located in the database directory **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts**.
3. Click **Open**.
4. We now need to remove some code in the `btnExecute_onclick()` method. The first task is to remove the prompt from the code. To do this just type `/**` in front of the prompt command. This will have the effect of commenting out the line completely, as shown below.

```
/**strCFile = prompt ("Enter the name of the collars file", "dhcollar");
```

5. Now we must add the code that will define the variable for the Browser Object as shown below. The `strCFile` variable already exists and the new variable will be placed immediately above it.

```
var oDmBrowser;
var strCFile;
```

6. We must now link that variable to the Object in the **Studio 3 Application Object**. This is shown below:

```
alert("This script Sorts the Collars File.");
// strCFile = prompt ("Enter the name of the collars file", "dhcollar");
oDmBrowser = oDmApp.ActiveProject.Browser;
```

7. Finally we need to access the `oDmBrowser` object and set the **TypeFilter** property, show the dialog to the user and then retrieve the **FileName** information. This is shown below, these commands should be placed straight after the creation of the object and before the confirm line:

```
oDmBrowser.TypeFilter = oScript.DmFileType.dmCollars;
oDmBrowser.Show(false);
strCFile = oDmBrowser.FileName;
```

8. The above lines allow the **TypeFilter** to be defined (in this case to the **dmCollars** type). This allows the browser to show Datamine files of the correct type when it is shown to the user. The **Show** method shows the dialog to the user. The assignment of the browser filename to the variable `strCFile` allows information to be obtained from the browser and into a local variable for use later on.
9. Use the find dialog (**Edit | Find...** from the menu) to search for `'_scr_SortCollars'` text, click the **Find** button. When the text is found change it to `'Sort Collars Browser'`. There should only be two occurrences of this text.

10. Save the file using **File | Save As...** from the menu and in the **Save As** dialog enter the name 'SortCollarsBrowser.htm' and click **Save**. The file extension should be defined for you, and the file should be saved to **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts**.
11. Leave this application open with the loaded file in case any changes need to be made.
12. Select **Tools | Scripting | Run Script...** from the main menu.
13. In the **Browsing for script file...** dialog select the *SortCollarsBrowser.htm* file.
14. In the **Customization** window note that the script is loaded and shows the 'Sort Collars Browser' text next to the CAE Mining logo.
15. Clear the **Command** window by right-clicking in the window and selecting **Clear** from the menu.
16. Execute the script.
17. The popup alert message is shown to the user. Click **OK**.
18. Select the *dhCollar* file from the Browser window and click **OK**.
19. Click **OK** to the confirmation message and view the output in the **Command** window.



We have used a **TypeFilter** of 'dmCollars' in the above example. There are many more filter types that can be used. If the file type is not known then the type **dmAny** should be used.

Model Prototype Script

The example script *_scr_Example Model Prototype.htm* (with a standard installation, this file is found under **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts**) provides an interface for defining the parameters of a model prototype. It then creates a wireframe model around the outside of the prototype model, and loads the wireframe into the visualizer. It uses both database and **Studio 3 commands**.

The script interface (with suitable data) looks like the following:

Define Model Parameters			
Output Model	X	Y	Z
protomod <input type="button" value="Browse ..."/>			
Origin	<input type="text" value="2100"/>	<input type="text" value="4500"/>	<input type="text" value="300"/>
Cell Size	<input type="text" value="10"/>	<input type="text" value="10"/>	<input type="text" value="5"/>
No. of Cells	<input type="text" value="20"/>	<input type="text" value="24"/>	<input type="text" value="22"/>
Maximum	<input type="text" value="2300"/>	<input type="text" value="4740"/>	<input type="text" value="410"/>
<input checked="" type="checkbox"/> Allow sub-cells			
<input type="button" value="OK"/>			

The script allows the user to browse for an *Outout model* name or supply a name in the text box and it allows for the input of X, Y and Z parameters.

HTML Objects

The HTML objects that are used are:

- Text boxes for all the XYZ data inputs and model filename.
- A check box for deciding whether to allow sub-cells or not.
- A push button.

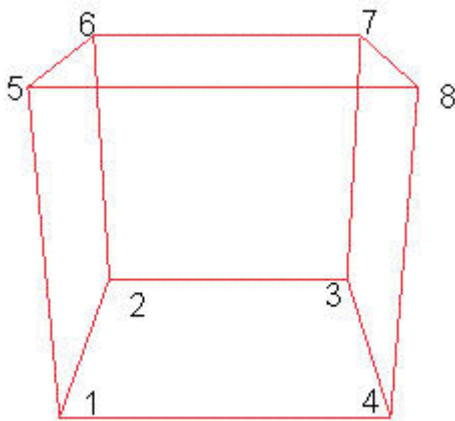
Studio Application Object Methods

The **Studio 3 Application Object Model** calls that are being made are:

- A call to **ActiveProject.Browser** to display the file browser.
- Many calls to **ParseCommand** to carry out the commands for creating and displaying the prototype.
- Calls to create and initialise the Application model object.

Creating the Wireframe

The wireframe is created by defining a closed string around each of the six faces of the rectangular model and end-linking them. The corners of the model are numbered as illustrated below:



The wireframe outline is loaded into the **Design** and **Visualiser** windows after the model has been created.

Points to Note: If you record a database command such as **PROTOM** that has interactive responses then the responses will be recorded in single quotes. These single quotes are optional and can be removed. However you must include a space between each interactive response as is illustrated in the script. A space is inserted into a character string as " ".



The text boxes that *represent Maximum X, Maximum Y and Maximum Z* have been write-protected and shaded. No manual input will be accepted into these text boxes.

Accessing Records and Fields

This example script *_scr_Example Record and Field.htm* (found with a standard installation at **C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts**) provides an interface for accessing the value of a field in a selected record from a Datamine file.

The script interface looks like the following:

Access fields and records

File

Record

Field

Value

HTML Objects

The HTML objects that are used are:

- Text boxes for the filename, record, field and value information.
- A browse button for selecting a file name.
- Navigation buttons to browse through the record and field information of the Datamine file.

Studio Application Object Methods

The **Studio 3 Application Object Model** calls that are being made are.

- A call to **ActiveProject.Browser** to display the file browser.
- A link to the **DmFile.DmTableADO** Object which is an interface for an ActiveX Data Object (ADO) record set on the open file. This object will provide all the information on the data within the file and for navigation through the file. These properties and functions include **MoveFirst**, **MoveLast**, **MoveNext**, **EOF**, **BOF**, **FieldCount**, **GetCurrentRow**, **GetRowCount**, **GetFieldName** and **GetColumn**.
- Calls to create and initialise the Application model object.



For further information on ADO refer to:

<http://msdn.microsoft.com/data/default.aspx>.

The Studio OLEDB provider currently allows reading and updating existing records. It does not allow creating new records or new files.

ExecuteCommand vs. ParseCommand

In the previous examples all Studio 3 *processes* and *commands* have been accessed through the **ParseCommand** method of the **Studio 3 Application Object**. For example, with the following command:

```
oDmApp.ParseCommand("protom &OUT=" + tbModel.value + "@ROTMOD=0" +  
" N " + subcells +  
" " + tbOriginX.value +  
" " + tbOriginY.value +  
" " + tbOriginZ.value +  
" " + tbCellSizeX.value +  
" " + tbCellSizeY.value +  
" " + tbCellSizeZ.value +  
" " + tbNumCellsX.value +  
" " + tbNumCellsY.value +  
" " + tbNumCellsZ.value + " ");
```

In each case the full set of files, fields, parameters and interactive responses must be included in the string that is sent by the command method to the **Studio Command Manager**.

If you want to invoke a command from a script, and you want the interactive responses to be properly interactive rather than pre-specified, then you must use the **ExecuteCommand** method instead of the **ParseCommand** method. For example:

```
oDmApp.ExecuteCommand("protom");
```

In this simple example you will be prompted interactively for the model origin, cell sizes etc. The script *_scr_Example Model Prototype Interactive.htm* can be run, and this file is found (with a standard installation) at

C:\Database\DMTutorials\Projects\S3ScriptTut\Scripts.

The **ExecuteCommand** method is particularly effective when commands such as 'zoom-all' and 'new-string' can be used from a script.

This concludes the Studio 3 Scripting User Guide.



8585 Cote-de-Liesse

Saint-Laurent, Quebec

H4T 1G8

Canada

Tel: +1 514 341 2000 ext2404

www.cae.com/mining