maxSCRIPT Script Language Programmer's Reference and User's Guide

277588 Rev. C

Refer to this publication for complete and accurate information that helps you better operate and service Metso Automation MAX Controls equipment. Your comments and suggestions are welcome.

Metso Automation MAX Controls
1180 Church Road
Lansdale, PA 19446
Attention: Manager, Technical Publications

Contents

PREFACE III
How This Book Is Organized...iii
Audience iii
Using Online Help iii
Using Other Programming Facilities...iii

CHAPTER 1 1-1

Introduction... 1-1
Overview..1-1
Creating a Display Containing Script...1-1
Creating Script Files...1-2
Developing Script Using maxSCRIPT...1-2
Using Dimension Statements 1-3
Using Local Variables..1-3
Reading and Writing from the Software Backplane...1-4
Performing External Writes 1-4
Interacting with Other maxVUE Controls..1-5
Writing Script Logic 1-6
Using Let and If, Else Commands..1-6
Using the Bitmap Command 1-7
Using Functions 1-7
Using Special Variables..1-8
Placing Scripts in Groups..1-8
Making Scripts Portable Using Parameterization 1-8
Specifying a Percent (%) Character 1-10
Putting It All Together..1-10
Reviewing maxSCRIPT Error Messages 1-10
Writes to Constants Are Prevented 1-11
Using the maxSCRIPT Editor...1-11
Creating New Files and Projects...1-12
Working with Projects 1-12
Renaming the Project..1-12
Adding Files..1-13
Renaming Files 1-13
Editing Files..1-14
Enabling Standard Editor Options 1-14
Using Auto Complete..1-15
Using Line Numbering..1-16
Using Line Highlighting...1-16
Using Syntax Highlighting...1-16
Formatting Files..1-17
Commenting and Uncommenting Text 1-17

Indenting and Outdenting Text 1-17
Toggling between Upper and Lower Case 1-18

CHAPTER 2........ 2-1

Creating maxSCRIPTs Using Simple Examples........ 2-1
Getting Started 2-1
Creating Points to be used in the Following Exercises........ 2-1
Creating and placing a maxSCRIPT Control 2-2
Exercise 1: Creating a Three-State Valve Condition Using Script........ 2-4
Testing Exercise 1 in Runtime Mode 2-6
Exercise 2: Creating a maxSCRIPT Control Using Parameters 2-6
Testing Exercise 2 in Runtime Mode 2-8
Exercise 3: Creating a Navigation Button Using Script 2-8
Testing Exercise 3 in Runtime Mode 2-10
Studying maxSCRIPT Examples 2-10
Example 1: Calculate an Average, Trigger an Alarm Indicator 2-11
Example 2: Working with Text Strings........ 2-12
Example 3: Hiding a Button in a Pop-up Display 2-13
Exercise 4: Using Flashing Bitmaps........ 2-14

CHAPTER 3........ 3-1

maxSCRIPT Script Language Reference Guide 3-1
Overview........ 3-1
Using Dimension Statements 3-1
Creating Local Variables........ 3-1
Creating Objects in the Local Status Server........ 3-2
Performing Demand Writes Using Command Statements........ 3-3
Performing External Writes 3-4
Subscribing to Keyboard Groups Using _Keyboard.respool 3-5
Interacting with Other maxVUE Controls........ 3-7
Using maxSCRIPT Commands........ 3-9
Using Let X = Y Expressions........ 3-9
Using If, Else, Else If, EndIf Expressions........ 3-10
Using the Bitmap Command 3-11
Using Flashing Bitmap Command........ 3-11
Using the Play Command........ 3-12
Using the ToolTip Command........ 3-12
Using the Run Command 3-12
Using Special Variables 3-13
Using the Display Variable 3-13
Using the Mouse Clicks Variable........ 3-13
Using the Error Variable 3-13
Using Variables Mouserow/ Mousecol to Locate Mouse Cursor........ 3-14
Using Special Functions........ 3-14
Using the Absolute Value Function........ 3-15
Using the ASCII Character Function........ 3-15
Using the ASK Function 3-15
Using the AVI Function 3-17
Using the Expand Function 3-18
Using the Logical Negation Function........ 3-18
Using the Objmap Function to Access the TypeMap 3-18

Using the P_CONTROL and P_DETAIL Functions 3-20
Using the QUA Function..3-21
Using the RGB Function..3-22
Using the SendMsg Function 3-23
Using the Service Function 3-23
Using the TRANS Function 3-23
Using the Value Function...3-23
Creating Unique Output Names 3-24
Changing maxSCRIPT Execution Time 3-24
Annotating Script with Comment Text 3-24

Preface

How This Book Is Organized

maxSCRIPT is an OCX Control that lets you design your own custom maxVUE display applications using an easy-to-learn, Basic-like script language. This publication introduces you to maxSCRIPT programming concepts and presents you with a script language reference guide located in Chapter 3. If you are not familiar with maxSCRIPT, refer to Chapter 1 for an introduction and description of the control and the associated script language. Refer to Chapter 2 to review tutorial exercises and real examples of maxSCRIPT scripts authored by Metso Automation MAX Controls engineers.

Audience

This publication is intended for process control and software engineers and programmers involved with the design and development of human-machine interface applications in a control environment.

Using Online Help

Many of the concepts noted in this publication, such as maxVUE groups and parameterization, are explained more fully in maxVUE online help. Press the F1 key to access help.

Using Other Programming Facilities

maxSCRIPT is one of several controls that add programming capability to maxVUE. Other facilities consist of:

Wrapper Calculations — The easiest to implement; use this to place a one-line calculation most places where a Software Backplane (SBP) identifier is used.

Hidden Logic — This has full Visual Basic capability, but in addition to maxVUE, you must make a separate purchase of the Visual Basic tools and have some knowledge of the Visual Basic environment.

You may also add Visual Basic and C + + controls to maxVUE to add full-blown programming capabilities to maxVUE, however, these options present some obvious disadvantages.

VB Control	This option presents you with a good deal of presentation flexibility but requires the separate purchase of VB Professional Version and the services of a VB programmer. It cannot take full advantage of parameterization.
C++ Control	Requires a professional C++ programmer.

Chapter 1

Introduction

Overview

maxSCRIPT is an OCX Control that helps you design a custom display within the maxVUE environment. Use maxSCRIPT for screen design applications to incorporate customized logic not available from any of the other existing standard controls.

maxSCRIPT uses a simplified Basic-like language that you can learn quickly to create logical statements and perform simple calculations. One or more instances of this control may be placed onto any maxVUE display.

Use the control to create local data and perform simple calculations using live data. The results of the calculations can be displayed on the associated display.

maxSCRIPT is an ideal way to create screen animations, such as an icon that changes color based on values obtained from multiple comparative inputs. For instance, use maxSCRIPT to create a pump icon. The valve on the pump may then be programmed to change color based on a given state, such as on, off, idle, broken, etc.

Color changes can be applied to other controls. For example, assume your process contains a tank that is filled with a reagent that must be heated to 1000 degrees. Script logic could be used to change the color of the graphic every 250 degrees. Logic could be set up as follows:

Degrees	**Color of Tank**
0 - 250	Blue
251 - 500	Yellow
501 - 750	Cyan
751 - 1000	Red

Creating a Display Containing Script

To create a display containing maxSCRIPT expressions and commands:

- Use the mouse to create and place the maxSCRIPT control object on your display. You may scale the control to any size including full screen.

- Access the Script property page to create or edit the script file in a maxSCRIPT Editor session. Use the Editor to create simple text files with a default extension of *.mxs*.

Note: the maxSCRIPT Editor, an integrated text editor, simplifies script editing and debugging. Scripts may also be edited in a simple text editor, such as Notepad, as long as the result is a simple ASCII text file (no formatting characters).

- Save the script file to a default script path, such as c:\custom\database\scripts.

An additional control property page called Default lets you define a bitmap that you want to appear in the maxSCRIPT window to replace the standard maxSCRIPT icon.

Similar to other controls, property pages are available to define the maxSCRIPT control as a navigation button (Display Navigation Page), to define borders for the button (Border Page), and to define window sizing and placement (Drawing Options).

Creating Script Files

Script files should have the extension .mxs. Note the control contains only a reference to a script file. It is common to reuse script files many times even on the same display.

A script may be stored with a display if used only by that display. When you intend to use scripts multiple times, they should be kept in the common area c:\custom\database\scripts.

maxSCRIPT is compiled in order to improve performance. When a script is executed for the first time by maxVUE Runtime, it is automatically compiled into a new file that is smaller and executes faster. The new file, which is located in the same folder as the text script file, has the extension “.mxo”.

One Caution: if you use Notepad or another text editor to edit a script outside of the maxVUE Editor, the old .mxo file will not be updated (the maxSCRIPT Editor automatically deletes the .mxo file for you if you make an edit). You can simply delete the .mxo file to cause a compilation to take place the next time that the script is executed.

Developing Script Using maxSCRIPT

Scripts associated with a particular presentation are typically small in size. You may place as many scripts within a graphics presentation as needed using multiple maxSCRIPT controls. There is one script per control.

Note: scripts have no size limits (other than any editing utility size limitations). Keep in mind, however, that as the script is compiled when the display is called up, large scripts may effect display performance.

Scripts consist of dimension statements, commands, variables, and special functions, and other elements, all of which are listed in Chapter 3, "maxSCRIPT Script Language Reference Guide."

Using Dimension Statements

Dimension statements, which will appear in virtually all scripts that you create, define names (aliases) that allow you to read data from and perform writes to local variables and to locations within the Software Backplane.

You may use dimension statements to:

- Create named objects in the Local Status Server to output information to other Software Backplane users
- Interact with other controls
- Define external values to be written to, such as an existing point in the Local Status Server that is preserved even when you close a maxVUE session.
- create local variables to be use exclusively within this program

Using Local Variables

The dimension syntax for a local variable is written as:

Dim <alias> as <type>

Where:

<alias> any string that uniquely identifies this value in the rest of the script

<type> can be any of the following:

dbl	Double precision, floating point integer
int	integer value
str	String characters such as a description or tagname

Example

```
Dim x as dbl
Dim x as int
Dim x as str
```

Reading and Writing from the Software Backplane

To read and write to values from the Software Backplane use the following syntax:

Dim <alias> as <type>(<service>.<extended member>)

Where:

<alias> any string that uniquely identifies this value in the rest of the script
<service> any valid SBP service name
<extended member> any valid SBP extension
<type> consists of the following in and out types

xxxIn types refer to an already existing variable, which is to be used in a calculation.

DblIn – double in	Double precision, floating point integer; typically a variable used in a calculation.
IntIn – integer in	Integer variable
StrIn – string in	String characters such as a description or tagname

xxxOut types create (and delete when the screen closes) an LSS object for output (as well as input if necessary) so that other controls can access the calculated results. A <service> is created for the purpose of housing outputs in the Local Status Server. The same service can (and should) be used for multiple members. The members can be of all different types.

DblOut – double out	Double precision, floating point integer; an output to be created so that controls can access calculated results
IntOut – integer out	Integer variable
StrOut – string out	String characters such as a description or tagname

Examples:

```
dim xyz as dblOut(b.condou)
dim i12 as intOut(b.conint)
dim txt as strOut(b.contxt)

dim asaw as dblIn(saw.ao)
dim lss_service_count as intIn(_lss.numsvc)
dim now as strIn(_lss.time)
```

Performing External Writes

To access an existing object as an output, write the dimension statement as *dimx*.

Example:

```
dimx s as strout(_sel_pt.tagname)
```

This lets you write to existing software backplane service members, such as DPU point (service) attributes (members) and LSS service members, such as _sel_pt.

The function of the *x* is to prevent the script from creating an _lss variable of the name (<external name>.<external attribute>), which is deleted when the script is terminated.

DIMX is only functional for dimension statements using out types as shown in the following:

```
dimx <name> intout(<external name>.<external attribute>)
dimx <name> dblout(<external name>.<external attribute>)
dimx <name> strout(<external name>.<external attribute>)
```

Example

```
Dimx selpt as strout(_sel_pt.tagname)
let selpt = “fic101”
```

Interacting with Other maxVUE Controls

To interact with other controls on a given maxVUE display, use the following syntax:

Dim <name> as <xxx>ctlout(<object name>.<parameter>

Where:

<name> is the alias for this parameter within maxSCRIPT
<xxx> is replaced by Dbl, Int or Str; note that the out types include the characters *ctl*, indicating the script will interact with other controls. For example, Dbl***ctl***out.
<object name> is the name of the object as seen and set in the **Layers** button of the maxVUE Editor
<parameter> is from the list:

x - the x position for percent of screen
y - the y position percent of screen
width - the width of the object
height - the height of the object
color - the color of the object
linecolor - the linecolor of the object

Your script should include one of the following out types

Dblctlout; double integer out to control	Double precision, floating point integer an output to be created so that controls

	can access calculated results.
Intctlout: Integer out to control	Integer value
Strctlout; string value out to control	String characters such as a description, or tagname

Example:

```
Dim x as intctlout(object.x)
Dim y as intctlout(object.y)
Let y = 10
```

Writing Script Logic

Once you identify write and read operations (using dimension statements to obtain data), you may use commands combined with logical expressions to perform various operations that incorporate the data. maxSCRIPT uses the following commands, variables, and functions to create logical arguments:

Table 1-1. maxSCRIPT Commands, Variables, and Functions

Commands	Variables	Functions
Let	Display	ABS; Absolute Value
If, else, etc	Mouseclick	ASC; convert number to ASCII character
Bitmap/fbitmap	Mouserclick	AVI; specify and display frames
Play	Mousedclick	EXP; expand parameters in real time
Tooltip	Mouserow	NOT; Logical Negation
Run	Mousecol	OBJMAP; perform lookup in typemap database
	Error	QUA; checking variable quality
		SVC; select a service
		TRANS; enable dual language translator
		RGB; calculate a color
		VAL; convert ASCII string to double

The basic syntax of any logical expression is *Command* plus *argument*. An expression begins with one of the commands listed in the preceding table and uses variables, functions, or a combination of both as part of its argument. Refer to the following sections for specific examples of these three script elements.

Using Let and If, Else Commands

Use *Let* statements and *if, else* statements to create logical expressions and set up simple arithmetic operations. The syntax of an expression using *Let* is:

Let variables = expression

For example:

```
Let c = a + b.
```

The following script sample uses *let* and *if*, *else* expressions.

```
let output = ( Input1 + Input2 ) / 2

if ( AnyAlarm1 = 1 ) or (AnyAlarm2 = 1 )
     let AnyAlarm = 1
Else
     let Anyalarm = 0
EndIf
```

Using the Bitmap Command

maxSCRIPT also lets you display bitmaps using the **Bitmap** command. You may create a maxSCRIPT control incorporating multiple bitmaps that may represent, say, a state change in your process or various mode changes, or whatever your application may call for. Using *if*, *else* logic, the script may display the bitmap associated with a given state. An additional **fbitmap** option lets you create a flashing bitmap. The following sample script shows the Bitmap command and the path name of a bitmap:

```
Bitmap "C:\custom\displays\operating\common\bmp\s-misc-1.bmp"
```

See "Using maxSCRIPT Commands," in Chapter 3 for a listing of other available commands and their syntaxes.

Using Functions

Functions are so called because they are used as a function of a variable in an expression as in this syntax:

Let variable = function argument

For example:

```
Let x = ABS (4)
```

maxSCRIPT supports the functions listed in Table 1-1, which are also listed in Chapter 3, "Using Special Functions."

The following sample script includes the RGB function. In this example the RGB function changes the color of the caption text placed on a button based on the state of the input BExp1. When the if statement becomes true, the caption appears, assuming the color represented by "96,96,96". When the if statement is not true, the RGB function generates a null string representing no color, effectively canceling the button text.

```
If BExp1 = 1
```

```
     Let ButtonTxtColor1 = RGB "96,96,96"
Else
     Let ButtonTxtColor1 = RGB "0,0,0"
Endif
```

Using Special Variables

Variables, such as **Display**, let you open a display or pop-up, and **Mouseclick**, let you create a script control that responds to mouse clicks. The following sample script logic contains both:

```
If mousedclick
     Let mousedclick = 0
     let display = "popups\zpop-ct-dv2-o-s-c.mn
     dp1=%param1%"
endif
```

See "Using Special Variables," in Chapter 3 for a listing of all the available variables and their syntaxes.

Placing Scripts in Groups

You may place scripts in a group making it possible to reuse them multiple times in many other instances. A Group is a named collection of objects, such as drawing objects, ActiveX Controls, and other groups, that maxVUE treats as a single entity. Groups can be cut, copied, pasted, moved, resized, and deleted just like other objects. This makes it easy to reuse a drawing depicting some process many times in other displays. You may create group libraries containing grouped drawings that can be incorporated in other displays and modified to adapt it for other purposes.

Where scripts are included with a group, cutting and pasting the group will also take the script reference along. Note in this case, multiple groups would be using the same script. This is handy when you want to make global changes to script logic that affect all instances.

Making Scripts Portable Using Parameterization

To make maxSCRIPTs even more portable, you may create scripts using parameterization. With parameterization you may create displays and display groups containing maxSCRIPT controls and other standard controls that do not contain specific point references. Instead, point references are indicated by %paramn% or %dpn% where *n* is the parameter or display parameter number. Expansion is identical to that of any other control.

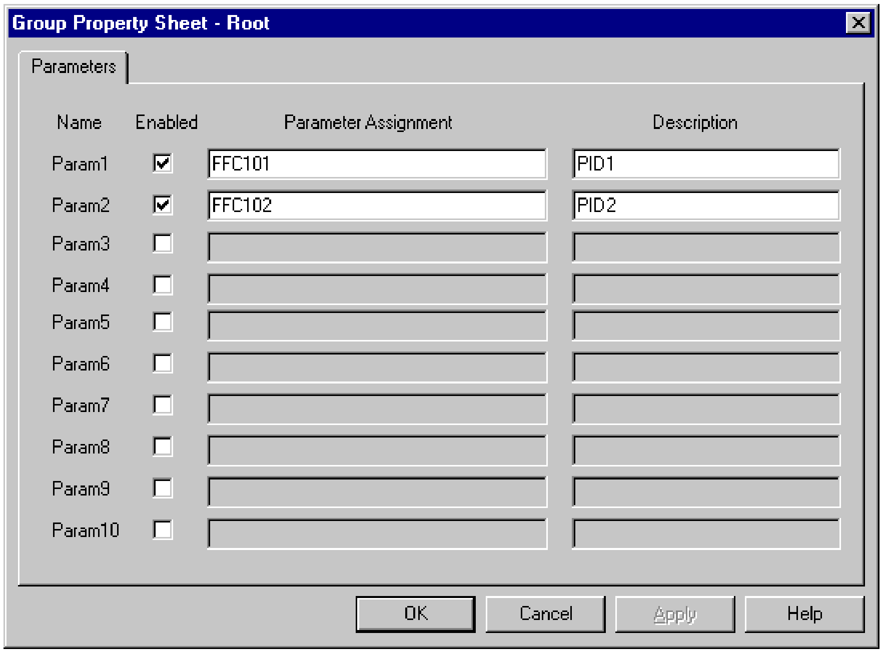
Example

```
Dim p1 as dblIn(%param1%)
Dim p2 as dblIn(%param2%.ao)
Dim out as dblOut(%param3%)
Let out = p1 + p2
```

Parameters, such as %Param1% or %DP1%, can then be defined differently for each display or display group.

You must include the control in a group – even if it contains only a single script item – to properly use parameters. If you edit the parameters of a control not in a group, you will be editing the one and only set of default parameters for the whole display.

Select **Edit Group Parameters** from the Group Menu to access the Parameters Property Sheet.



Name	Enabled	Parameter Assignment	Description
Param1	☑	FFC101	PID1
Param2	☑	FFC102	PID2
Param3	□		
Param4	□		
Param5	□		
Param6	□		
Param7	□		
Param8	□		
Param9	□		
Param10	□		

Enter the point name to be used with the Parameter in the Parameter Assignment field and a description for the parameter. A parameter may be defined to be another parameter such as %DP1%. At run-time, maxVUE will continue parsing the parameterized name until it has been resolved.

Now, you may create a display once and use it many times in many places. If you define subsystems using hierarchical identifiers, for instance, you will probably find yourself creating similar displays for multiple hierarchical groups.

You are now free to copy the display containing the parameter names. At runtime, maxVUE decodes the parameters it finds and inserts the actual values.

Note: by using parameters for the outputs of the script, the same parameter can be used for the script as is used for a Bar or a List Box insuring no mistakes.

Specifying a Percent (%) Character

Percents are part of the expansion of parameters - %param1% for example. As such they are special characters. If you actually want a percent (%), you must use %%. Example:

```
Dim x as dblin(test.op%%)
```

Putting It All Together

The following shows a complete sample script incorporating many of the elements introduced in the proceeding discussion. This script will cause an on-screen alarm indicator to be set based on the state of two inputs. For a detailed discussion of this script, refer to Chapter 2, "Example 1: Calculate an Average, Trigger an Alarm Indicator."

```
Bitmap "C:\custom\displays\operating\common\bmp\s-misc-1.bmp"


Dim Input1 as intin(%param1%.out)
Dim Input2 as intin(%param2%.out)
Dim Output as intout(%param1%_avg.out)

Dim AnyAlarm1 as intin(%param1%.Anyalarm)
Dim AnyAlarm2 as intin(%param2%.Anyalarm)
Dim AnyAlarm as intout(%param1%_avg.Anyalarm)

let output = ( Input1 + Input2 ) / 2

if ( AnyAlarm1 = 1 ) or (AnyAlarm2 = 1 )
     let AnyAlarm = 1
Else
     let Anyalarm = 0
EndIf
```

Reviewing maxSCRIPT Error Messages

maxSCRIPT produces two types of errors, run-time and compile. The script is compiled when it is loaded. Syntax is partially checked. Run-time errors are mostly "tag or attribute not found" types of errors. To view errors in the maxVUE editor, right click on the control and select **Control Properties** from the menu to bring up the Scripts property page. View errors from this page.

Writes to Constants Are Prevented

maxVUE includes diagnostic capabilities to prevent writes to constant data. The following will result in an error:

```
dim s as strin(_lss.time)
let s = "now"
```

This generates the following error message. The *3* indicates that the error was detected at line three.

```
Write to a constant (3)
```

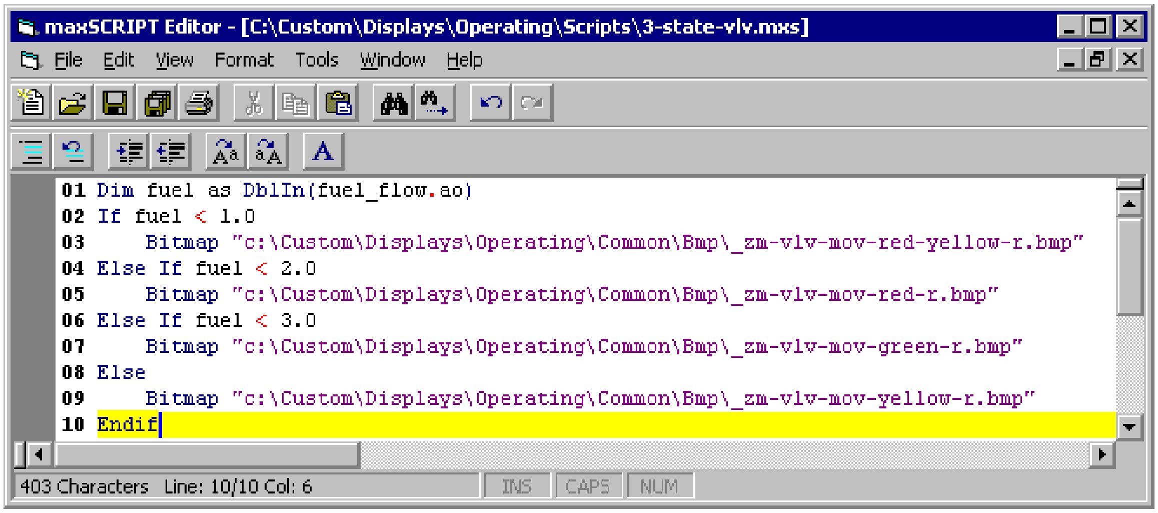
In another script example, the following contains a logic error that would yield the erroneous result *i = 2*. maxSCRIPT diagnostic capabilities prevent this from occurring.

```
dim i as int
let 1 = 2
let i = 1
```

Using the maxSCRIPT Editor

Use the maxSCRIPT Editor you to view, create or edit maxSCRIPT (*.mxs) files in a multiple document environment. The integrated maxVUE Editor, similar to any text or word processing program, simplifies maxSCRIPT editing and debugging as you create process displays.

When you place a maxSCRIPT instance on a maxVUE display and create a script file, the following editing window appears: Most of the basic functions can be accessed through a menu, a toolbar button or a shortcut key.


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\3-state-vlv.mxs]
File
Edit
View
Format
Tools
Window
Help
403 Characters Line: 10/10 Col: 6
INS
CAPS
NUM

```
Dim fuel as DblIn(fuel_flow.ao)
If fuel < 1.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-red-yellow-r.bmp"
Else If fuel < 2.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-red-r.bmp"
Else If fuel < 3.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-green-r.bmp"
Else
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-yellow-r.bmp"
Endif
```

Creating New Files and Projects

maxSCRIPT Editor allows you to create individual maxSCRIPT files or a maxSCRIPT project. Create a maxSCRIPT project to group together common or relevant files. See "Working with Projects," the next section.

To create a new maxSCRIPT file:

From the File menu, point to New and click maxSCRIPT file.

Alternatively, click the New icon on the toolbar or press <CTRL> + N.

To create a new project click 'New' then 'maxSCRIPT Project' under the File menu or the <CTRL> + J shortcut key.

Working with Projects

To group related maxSCRIPT files in one place, create a project file.

To create a project file:

From the File menu, point to New and click Project to access the Create Project dialog.

Enter a file name for the project, select a directory location, and click Save. When you save the file, the Editor adds a *.msp to the file name.

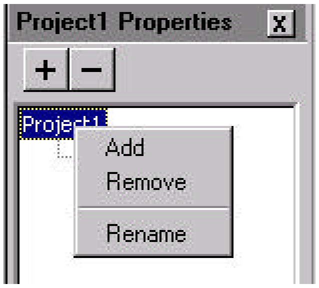
When you create the project file, the Editor opens a properties window containing a Windows Explorer style directory tree showing all the files in the current project.

The files can be given a name to identify themselves in the project. Any changes to the project will not be saved until Save Project is clicked under the File menu.

Renaming the Project

To change the name of the current project:

Select the root file name, the topmost node in the project window, right click and select Rename from the popup menu.



Renaming the project gives the project a name to identify itself and does not change the actual filename.

Adding Files

To add a file to the project

Right-click anywhere in the project window and click Add, or click the Add + button on the project window toolbar.

Removing Files

To remove a file from the current project:

right click on the file you wish to remove and click Remove, or click the Remove − button on the project window toolbar.

Renaming Files

To change the name of a file in the current project, select the file you wish to rename, right click and select Rename from the popup menu. Renaming the file simply changes the name to identify itself in the project and does not change the actual filename.

Creating New Files and Projects

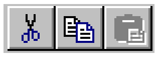
maxSCRIPT Editor allows you to create individual maxSCRIPT files or a maxSCRIPT project that can be used to group common or relevant files in an orderly manner.

Creating New Files To create a new maxSCRIPT file simply navigate the File menu to New then maxSCRIPT File. You can also click the New button on the toolbar or use the <CTRL> + N shortcut key.

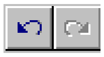
Creating New ProjectsA maxSCRIPT project helps the user to group relevant files in an easily accessible fashion. To create a new project click

New then maxSCRIPT Project under the File menu or the <CTRL> + J shortcut key.

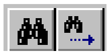
Editing Files



Cut, Copy, Paste Cut, copy, and paste functions can be accessed from the Edit menu or by clicking their icons on the toolbar.



Undo and Redo Undo and redo can be accessed from the Edit menu or by clicking their icons on the toolbar. Undo can also be accessed by using the **<CTRL> + Z** shortcut key. You can undo and redo actions an unlimited amount of times.



Find, Find Next, and Replace

Find, find next and replace functions can be accessed from the Edit menu, clicking their icons on the toolbar or using their <CTRL> + F, F3, and <CTRL> + H shortcut keys respectively.

Selecting all text can be accessed from the Edit menu or by using the <CTRL> + A shortcut key.

Enabling Standard Editor Options

The maxSCRIPT Editor includes standard features that can be enabled or disabled from the Options menu accessible from the Tools menu:

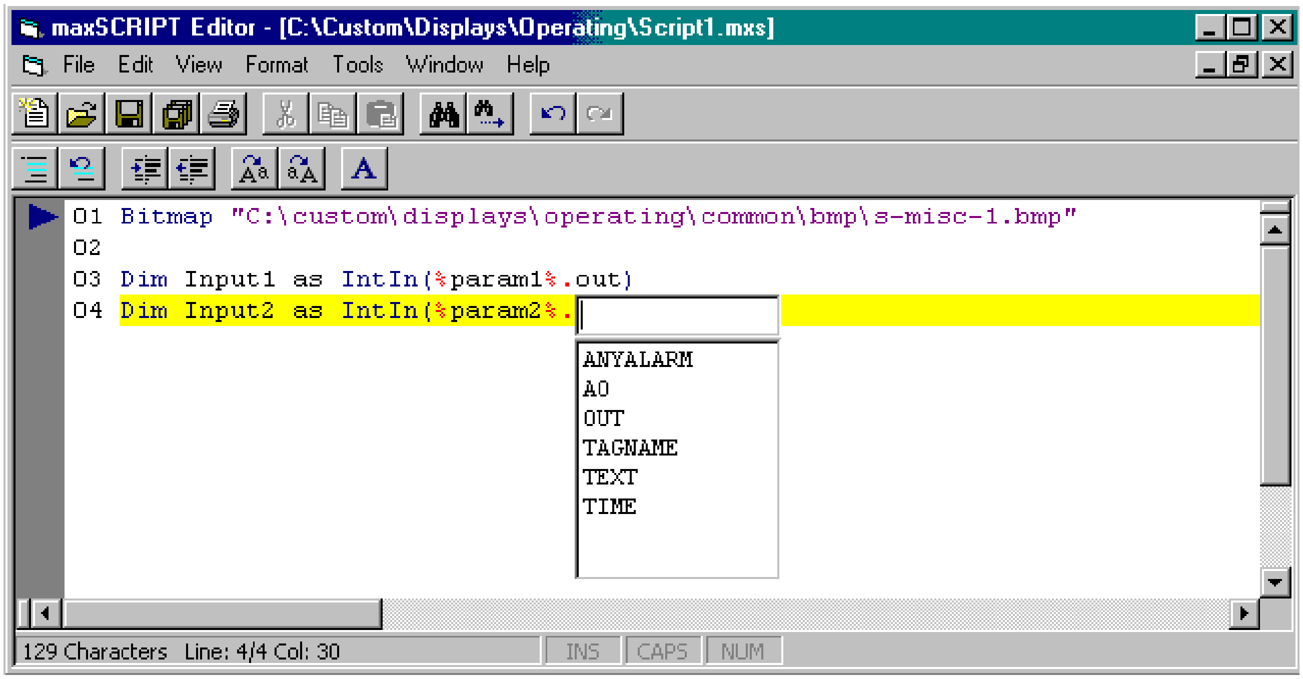
Auto complete

Line numbering

Line highlighting

Syntax highlighting

All of these features can be seen in the following picture of an edit window:


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Script1.mxs]
File Edit View Format Tools Window Help
ANYALARM
AO
OUT
TAGNAME
TEXT
TIME
129 Characters Line: 4/4 Col: 30
INS
CAPS
NUM

```
Bitmap "C:\custom\displays\operating\common\bmp\s-misc-1.bmp"

Dim Input1 as IntIn(%param1%.out)
Dim Input2 as IntIn(%param2%.
```

Using Auto Complete

Use the auto complete feature to select commonly used attributes from a pop-up list as you create script. The feature anticipates the attribute you intend to type and automatically completes the word.

To use auto complete:

Select Options from the Tools menu to enable auto complete, if the feature is not already enabled. From the Options dialog, check Auto Complete to enable it. Uncheck the checkbox to disable the feature.

Inside the maxSCRIPT Editor window, press the period key <.> to access a pop-up list of commonly used attributes.

As you continue to type as normal, the auto complete feature will select the word it thinks you are typing.

Press the [TAB] or [SPACE] key or click the selection to finish the word. Press the [ENTER] key to insert whatever word you have typed in the auto complete box without finishing the word.

You may customize the auto complete attributes list to suit your specific requirements.

To add attributes to the auto complete list:

Select MaxScriptEditor.txt from the Tools menu. When you edit the MaxScriptEditor.txt file it will be placed in C:\Custom\Displays\

Operating\Database\. The original MaxScriptEditor.txt file can be found in the C:\MCS\Setup\ directory and should not be edited.

Using Line Numbering

Line numbers permit you to quickly move to a specific area in a file or to debug an error from maxVUE.

To move directly to a specific line number:

Select Options from the Tools menu to enable line numbering, if the feature is not already enabled. From the Options dialog, check Line Numbering to enable it. Uncheck the checkbox to disable the feature.

Press <CTRL> + G. to access the Goto Line dialog box , enter a line number and click OK.

To change the appearance of line numbers:

Select Options from the Tools menu and under line numbering, check Bold or Italics.

Using Line Highlighting

Line highlighting can help to identify the current line being edited by coloring the current line yellow.

To enable or disable line highlighting:

Select Options from the Tools menu and check Line Highlighting to enable the feature or uncheck the checkbox to disable the feature.

Using Syntax Highlighting

Syntax Highlighting greatly reduces the chance of error by coloring keywords, operators and strings common to the maxSCRIPT language. Keywords are colored blue, operators red, and strings purple.

To enable or disable syntax highlighting:

Select Options from the Tools menu and check Syntax Highlighting to enable the feature or uncheck the checkbox to disable the feature.

Formatting Files

Commenting and Uncommenting Text

To annotate maxSCRIPT with free form comments, notes, special instructions, and so forth, begin each comment line with any of the following:

Rem
; Semicolon character
' apostrophe character

When maxSCRIPT is executed, any text beginning with the above characters is ignored.

To comment text automatically:

Place your cursor on the line to be commented and click the Comment button on the formatting toolbar.

You may also select Comment Block from the Format pull-down menu or press <CTRL> + I shortcut key.

To uncomment text automatically:

Place your cursor on the line to be uncommented and click the Uncomment button on the formatting toolbar.

You may also select Uncomment Block from the Format pull-down menu or press < <CTRL> + U shortcut key.

Indenting and Outdenting Text

To indent a line of text in a script file:

Place your cursor on the line you wish to indent and click the Indent button on the formatting toolbar.

You may also select Indent Text from the Format pull-down menu, or press the <Tab> key.

To outdent a line of text in a script file:

Place your cursor on the line you wish to outdent and click the Outdent button on the formatting toolbar.

You may also select Outdent Text from the Format pull-down menu, or press <SHIFT> + TAB.

Toggling between Upper and Lower Case

To change text from uppercase to lowercase and vice-versa navigate the Format menu, click the To Lowercase or To Uppercase buttons on the toolbar, or by use their <CTRL> + L and <CTRL> + K shortcut keys respectively.

Toggle Bookmark To toggle a bookmark at the current line click Toggle Bookmark from the Format menu. Bookmarks allow you to quickly access specific lines of text. You can make as many bookmarks you want in the opened file but will not be retained after closing and opening the file again. If you're using maxVUE Editor to create process graphics and a maxSCRIPT error is generated, the maxSCRIPT file will be opened in maxSCRIPT Editor with a bookmark at the line with the error.

Moving through Bookmarks: to move between bookmarks click Next Bookmark or Previous Bookmark from the Format menu or use the F2 and <SHIFT> + F2 shortcut keys respectively.

Clear Bookmark: to clear all the bookmarks in the current file click Clear Bookmarks from the Format

Shortcut Keys

maxSCRIPT Editor has many shortcut keys that make it easy to access common operations. The following table lists the default keystrokes and their associated actions.

New maxSCRIPT File	**Ctrl-N**
Open maxSCRIPT File	**Ctrl-O**
Save File	**Ctrl-S**
New Project	**Ctrl-J**
Open Project	**Ctrl-R**
Print	**Ctrl-P**
Cut	**Ctrl-X**
Copy	**Ctrl-C**
Paste	**Ctrl-V**
Select All	**Ctrl-A**
Select Line	**Ctrl-Alt-F8**
Undo	**Ctrl-Z**
Next Word	**Ctrl-RightArrow**
Previous Word	**Ctrl-LeftArrow**
Delete Next Word	**Ctrl-Delete**
Delete Previous Word	**Ctrl-Backspace**
Document Start	**Ctrl-Home**

Document End	**Ctrl-End**
Comment Block	**Ctrl-I**
Uncomment Block	**Ctrl-U**
Indent	**Tab**
Outdent	**Shift-Tab**
Convert to Lowercase	**Ctrl-L**
Convert to Uppercase	**Ctrl-K**
Create New Line (above)	**Ctrl-Shift-N**
Next Bookmark	**F2**
Previous Bookmark	**Shift-F2**
Find	**Ctrl-F**
Find Next	**F3**
Find Previous	**Shift-F3**
Replace	**Ctrl-H**
Goto Line	**Ctrl-G**
Goto Matching Brace	**Ctrl-]**
Set Repeat Count	**Ctrl-R**
Toggle Whitespace Display	**Ctrl-Alt-T**
Insert File Path from Dialog	**Ctrl-F1**
Scroll Window Down	**Ctrl-UpArrow**
Scroll Window Up	**Ctrl-DownArrow**
Scroll Window Left	**Ctrl-PageUp**
Scroll Window Right	**Ctrl-PageDown**
Help	**F1**

Note: Holding down shift when using any of the navigational shortcut keys will select any text that the cursor moves through.

Chapter 2

Creating maxSCRIPTs Using Simple Examples

Getting Started

To use the maxSCRIPT control effectively you need to become acquainted with the maxSCRIPT scripting language. This simple but powerful script language consists of a relatively small number of commands, special variables and functions that you may learn with ease. Script elements are all listed in detail in the maxSCRIPT Script Language Reference Guide in Chapter 3.

This chapter shows you how to create a few simple maxSCRIPT controls by example in three introductory exercises. The first exercise shows you how to display bitmaps based on the state of an analog output. A second exercise shows you how to implement a similar maxSCRIPT using parameterization. A third exercise demonstrates the use of the mouseclick variable.

The exercises are followed by four examples of script developed for real applications by Metso Automation MAX Controls engineers. Each example illustrates an aspect of script that you may implement when you compose your own script files.

Creating Points to be used in the Following Exercises

Before getting started, you must first create the temporary points, *fuel_flow.pid* and *air_flow.pid.* These points will be used as simulated controllers that you may manipulate throughout these exercises.

To create simulated points:

1. From the **Start** menu, point to **Programs, maxDNA, Utilities**, and click **TestSBP** to open the **TestSBP** utility dialog display.

2. Under Operation, select **Write**; under Write Type select **Empty**.

3. In the Identifier field, type the following:

   _lss.addsvc. fuel_flow.pid

4. Click the **Apply** button to add this simulated point to the Local Status Server.

After you add fuel_flow.pid, follow the same steps to add the second simulated point, air_flow.pid.

When you finish, you will have added two new services to the Local Status Server. To confirm that the services were added, open the LLS dialog and locate the two points you added under the service list. When you select any of the points, a list of default attributes appears under the Member list, including AO.

Note: when you close maxVUE and shutdown the maxSTATION, these two services will be deleted from the Local Status Server.

Creating and placing a maxSCRIPT Control

Before demonstrating maxSCRIPT script creation using these examples, lets get started with the basics, calling out the control.

To place a maxSCRIPT control in a maxVUE display:

1. Double click on the maxVUE Editor icon appearing on the Windows NT desktop to open the editor tool.

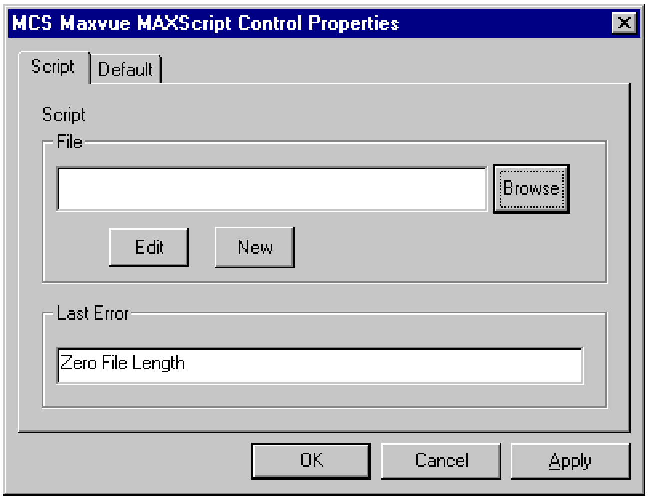
2. Click the **New** tool button to access the file directory dialog box and enter a file name, such as *mydisplay.mn*, or any name meaningful to you.

3. Click the **maxSCRIPT Control** tool button from the OLE Animator tool bar. Place the cursor on the desired location on the screen and click but do not release the button.

4. Drag the mouse to size the maxSCRIPT display window as desired and release the button when satisfied with the dimensions. Remember this will be the size and location of the bitmap window.

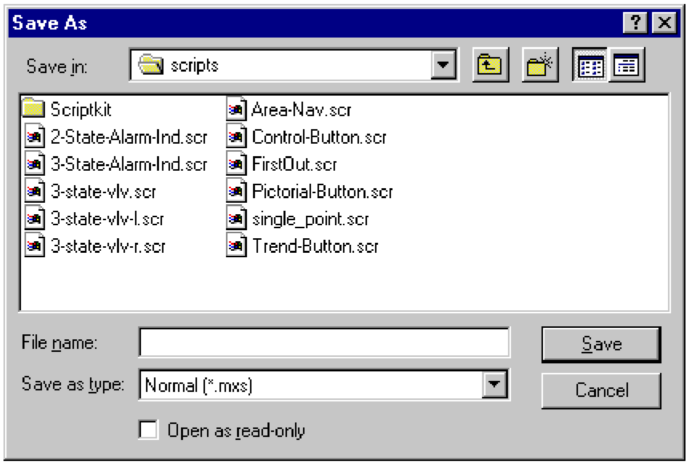
5. Click the **Select** tool button and click on the maxSCRIPT control you just created to select it.

6. Right-click to bring up the Control Menu pop-up and select **Control Properties**_from the menu to bring up the control's two property pages, **Script** and **Default**.



Last Error: Zero File Length

7. From the Script tab, click the **New** button to create a new script and enter an appropriate name for the script. Create a file location under the Custom directory such as c:\custom\database\scripts. Any additional scripts that you create should be saved to this location.



Save in: scripts

Scriptkit
2-State-Alarm-Ind.scr
3-State-Alarm-Ind.scr
3-state-vlv.scr
3-state-vlv-l.scr
3-state-vlv-r.scr
Area-Nav.scr
Control-Button.scr
FirstOut.scr
Pictorial-Button.scr
single_point.scr
Trend-Button.scr

File name:

Save as type: Normal (*.mxs)

☐ Open as read-only

8. Click the **Save** button to bring up the maxSCRIPT Editor.

9. From the maxSCRIPT Editor, enter the desired code and select **Save** from the File menu when you are done.

10. From the **Default** tab, enter an image file or background color that you want to appear in the control window to replace the standard icon that appears when you first call out a maxSCRIPT instance.

This summarizes all the steps required to create a maxSCRIPT control. Now you're ready to create scripts based on the examples outlined at the beginning of this chapter. Refer to the next sections.

Exercise 1: Creating a Three-State Valve Condition Using Script

In this first exercise, you will create a maxSCRIPT control that displays one of four bitmaps based on the state of an analog output. The bitmaps may consist of graphical representations of a pump valve. Each bitmap depicts an identical pump valve image, however, the fill colors vary from image to image. One bitmap may be red, another green, another yellow, and a fourth image may use a combination of red and yellow. Each color represents a different flow state. The following figure shows one version of the pump valve bitmap rendered in gray scale.

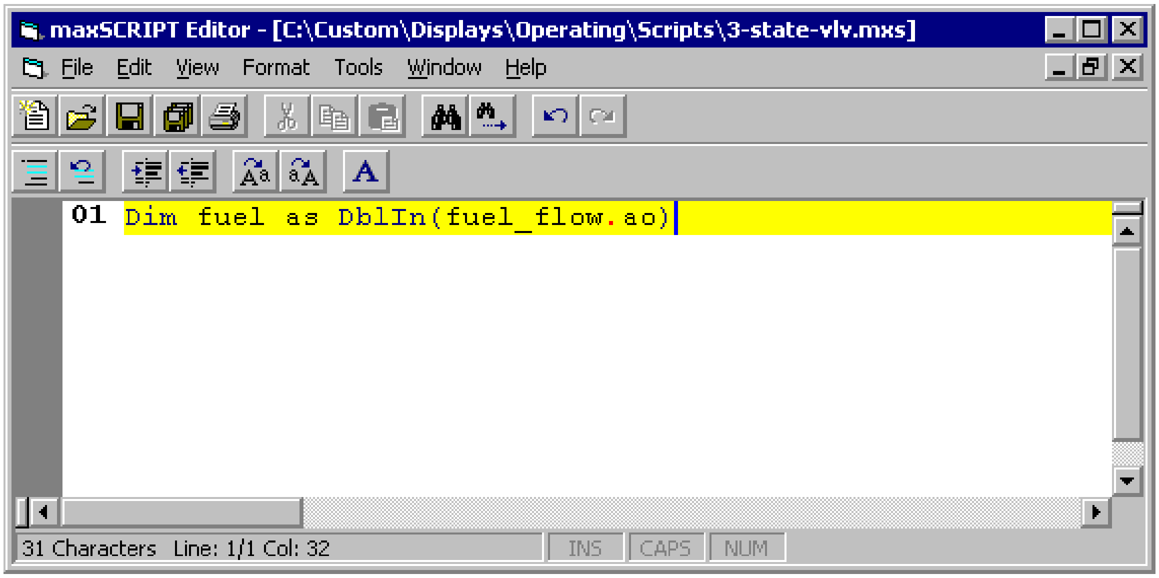
The script will consists of three common script language elements, the dimension statement, *If*, *Else* conditional statements, and the bitmap command. In addition to the maxSCRIPT pump control, you'll be creating another maxSCRIPT control containing a button that calls out the control.

Refer to the "maxSCRIPT Script Language Reference Guide" in Chapter 3 for a full listing of script expressions and commands and their syntaxes.

To create the control:

Follow the steps in the previous section, "Creating and Placing a maxSCRIPT Control," to call out the control and open a maxSCRIPT Editor session.

1. To begin the script enter the following dimension statement:



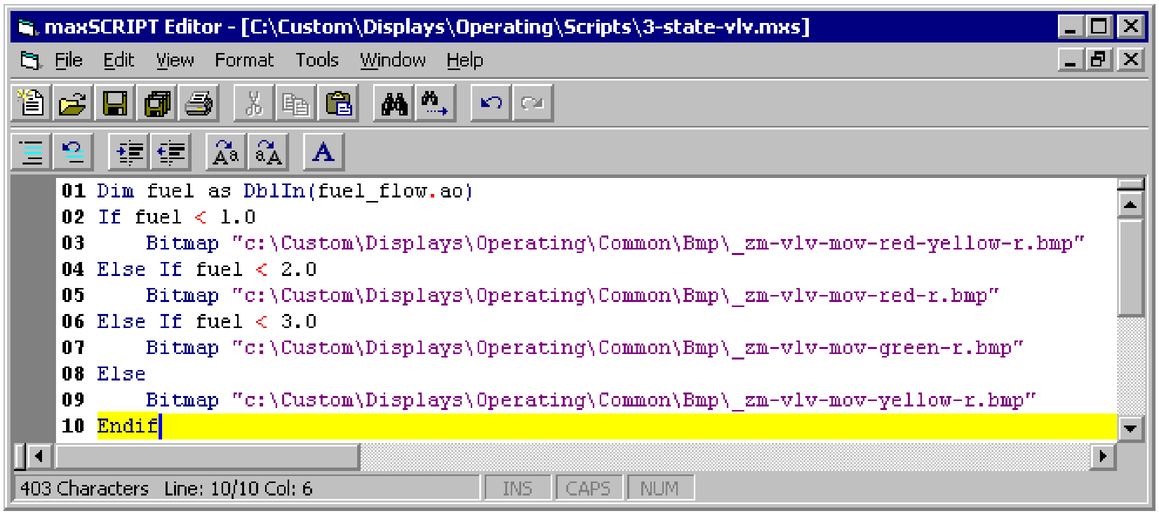
maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\3-state-vlv.mxs]
File
Edit
View
Format
Tools
Window
Help
INS
CAPS
NUM

```
Dim fuel as DblIn(fuel_flow.ao)
```

31 Characters Line: 1/1 Col: 32

In this dimension statement *fuel* is a double (floating) in from the tagname fuel_flow.ao.

2. The rest of the script uses *if*, *else* conditional expressions that incorporate bitmap commands as shown in the following figure:


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\3-state-vlv.mxs]
File Edit View Format Tools Window Help
403 Characters Line: 10/10 Col: 6
INS
CAPS
NUM

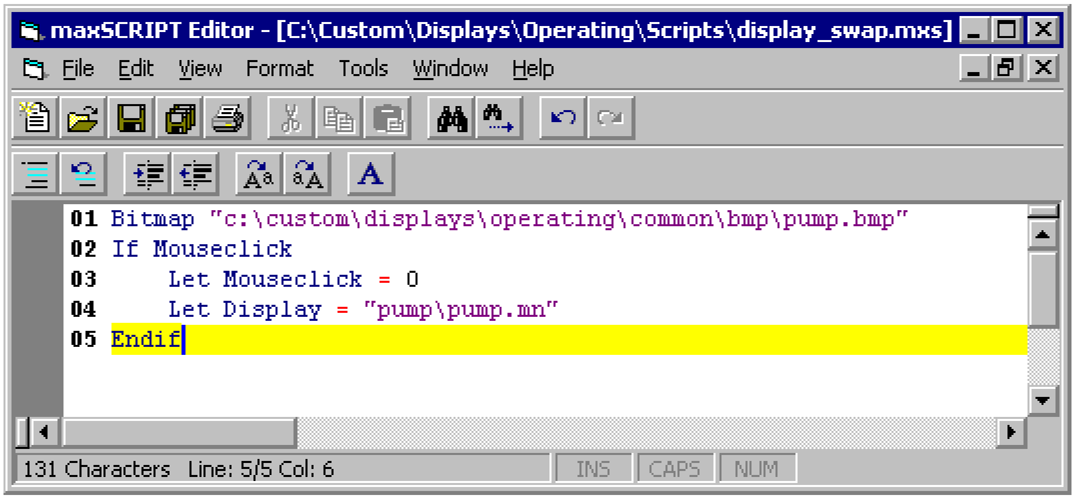
```
Dim fuel as DblIn(fuel_flow.ao)
If fuel < 1.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-red-yellow-r.bmp"
Else If fuel < 2.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-red-r.bmp"
Else If fuel < 3.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-green-r.bmp"
Else
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-yellow-r.bmp"
Endif
```

3. Save the script file using maxSCRIPT Editor menu options. In our example, the script is named *3-state-vlv.mxs*.

4. From the Controls Property Default Page select a default bitmap to display in the maxSCRIPT control window. This should be one of the bitmaps based on valve conditions.

5. Click on the **Save** tool button to save the changes to *mydisplay.mn*, or whatever name you choose.

   Create a screen button using a new instance of maxSCRIPT to display swap to *mydisplay.mn*, and close the maxVUE Editor. Type the following script to create a screen button in maxSCRIPT:


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\display_swap.mxs]
File Edit View Format Tools Window Help
131 Characters Line: 5/5 Col: 6
INS
CAPS
NUM

```
Bitmap "c:\custom\displays\operating\common\bmp\pump.bmp"
If Mouseclick
    Let Mouseclick = 0
    Let Display = "pump\pump.mn"
Endif
```

In this suggested script, the following line contains the path name of the target display:

```
Let display = "pump/pump.mn"
```

Substitute here the name of your target display containing the maxSCRIPT you created in this exercise.

Note: exercise 3 discusses the mouseclick variable at length.

Testing Exercise 1 in Runtime Mode

To see if the script you just created actually works, you must open it in maxVUE Runtime.

To test your script:

1. From the Windows NT desktop, double click on the maxVUE Runtime icon to invoke the application.

2. Click anywhere on the logo screen (but away from the maxVUE logo animation) to open the Main Menu display.

3. Click on the maxSCRIPT button you created to display swap to the *mydisplay.mn* display.

4. Select *fuel_flow* as the selected point and bring up the faceplate. Increase or decrease the AO to within the valve condition values specified in the maxSCRIPT. Are the Bitmaps changing based on the AO of the specified tag?

Exercise 2: Creating a maxSCRIPT Control Using Parameters

In this exercise, you'll create a maxSCRIPT control similar to the one you created in Exercise 1, only this time instead of using a specific point identifier, you will use parameters. As noted in Chapter 1, parameterization makes it possible to create a control once and reuse it again for other screen applications. Refer to "Making Scripts Portable Using Parameterization."

Before you attempt to perform this exercise, review Exercise 1.

To create a control using parameters:

1. From the maxVUE Editor open the display you created in Exercise 1.

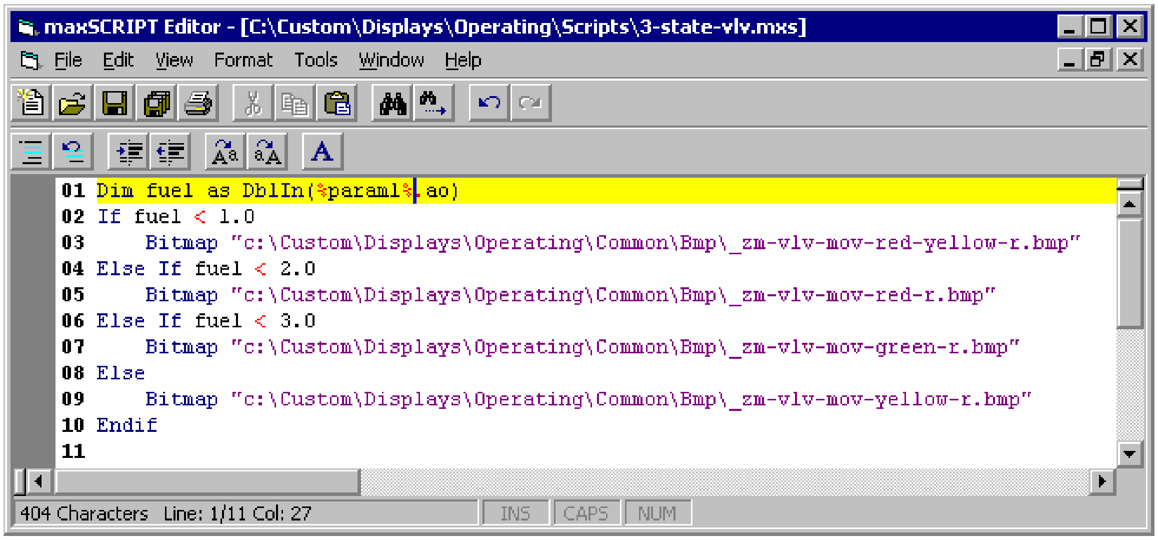
2. Click the ↖ **Select** tool button and click on the maxSCRIPT control you created in the previous exercise to select it.

3. Copy and paste the maxSCRIPT instance somewhere on the display.

4. Right-click to bring up Control Menu pop-up and select **Control Properties** from the menu to bring up the control's two property pages, Script and Default.

5. From the Script tab, click the **Edit** button to select the script for the 3-state valve condition you created earlier.

6. In the dimension statement, change *fuel* to *ao* as a double (floating) in from a group parameter instead of a specific point. Replace the point identifier (fuel_flow.ao) with (%Param1%.ao). Refer to the revised script in the following figure.


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\3-state-vlv.mxs]
File
Edit
View
Format
Tools
Window
Help
INS
CAPS
NUM

```
Dim fuel as DblIn(%param1%.ao)
If fuel < 1.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-red-yellow-r.bmp"
Else If fuel < 2.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-red-r.bmp"
Else If fuel < 3.0
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-green-r.bmp"
Else
    Bitmap "c:\Custom\Displays\Operating\Common\Bmp\_zm-vlv-mov-yellow-r.bmp"
Endif

```

404 Characters Line: 1/11 Col: 27

7. From the maxSCRIPT Editor File menu, save the script under a new name, such as *3-state-vlv-param*.

8. Click the **Select** tool button and click on the maxSCRIPT control you created. Right-click to bring up Control Menu pop-up and select **Group** from the context menu.

9. Select the **Group, Create New Group – From Selected Objects** option to make the maxSCRIPT instance a group.

10. Select the **Group, Edit Group Parameters** option and edit the Param1 assignment to be Fuel_flow.

11. Now create a second instance of the script for the point air_flow. Click the **Select** tool button and click on the maxSCRIPT control you created in this exercise to select it. Copy and paste the maxSCRIPT instance somewhere on the display.

12. Edit group parameter 1 of the copied script to air_flow.

13. Click the **Save** tool button to save the changes to your *mydisplay.mn* display and close the maxVUE Editor.

Testing Exercise 2 in Runtime Mode

To see if the script you just created actually works, you must open it in maxVUE Runtime.

To test your script:

1. From the Windows NT desktop, double click on the maxVUE Runtime icon to open the application.

2. Click anywhere on the logo screen (but away from the maxVUE logo animation) to open the Main Menu display.

3. Edit the screen button you created in the previous exercise to display swap to *mydisplay.mn*, and close the maxVUE Editor.

4. Select fuel_flow as the selected point and bring up the faceplate. Increase or decrease the AO to within the valve condition values specified in the maxSCRIPT. Now select and manipulate air_flow to view two instances of a script control operating independently while using the same script file. Are the Bitmaps changing based on the AO of the specified tag?

Exercise 3: Creating a Navigation Button Using Script

maxSCRIPT includes three mouse click variables that you may incorporate in a maxSCRIPT control with other script logic. Using a specific mouse click variable, you may design a screen navigation button that responds to a single left mouse click, single right mouse click or a double click. The button may use any background including bitmaps and be scaled to any size including full screen.

Refer to the "maxSCRIPT Script Language Reference Guide" in Chapter 3 for a listing of mouseclick variables.

To create script using a mouse click variable:

1. With the maxVUE Editor open on your display, click on the **New** tool button to access the file directory dialog box and enter a file name, such as *mydisplay_exercise3.mn* or any name of your choosing.

2. Click on the **maxSCRIPT Control** tool button from the OLE Animator tool bar. Place the cursor on the desired location on the screen and click but do not release the button.

3. Drag the mouse to size the maxSCRIPT display window as desired and release the button when satisfied with the dimensions. Remember this will be the size and location of the navigation button.

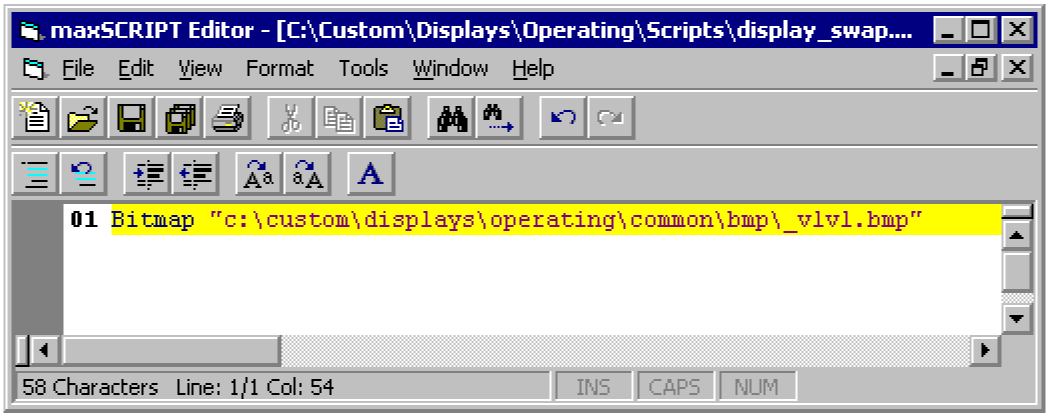
4. Click the **Select** tool button and click on the maxSCRIPT control you just created to select it.

5. Right-click to bring up Control Menu pop-up and select **Control Properties**_from the menu to bring up the control's two property pages, Script and Default.

6. From the Script tab, click the **New** button to create a new script and enter an appropriate name for the script, such as *display_swap.mxs.* Click the Save button to bring up the maxSCRIPT Editor.

7. From the maxSCRIPT Editor, enter the desired code and select **Save** from the File menu when you are done. Create a file location under the Custom directory such as c:\custom\database\scripts.

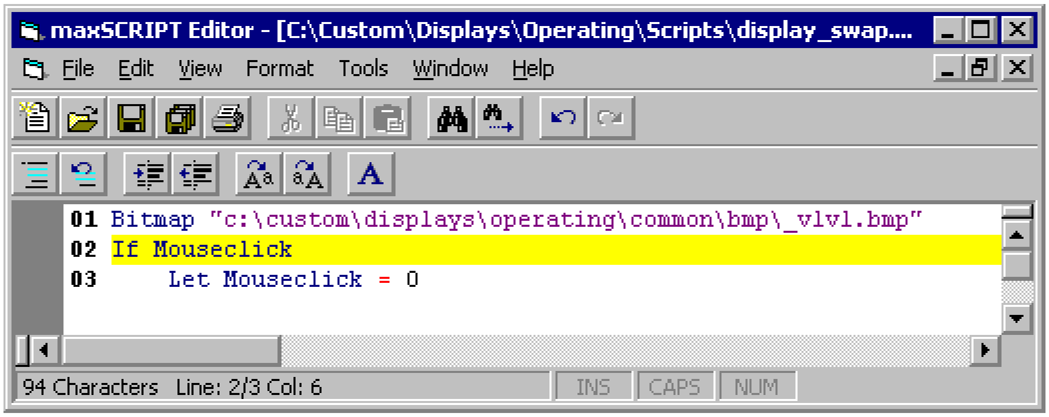
8. The first line of script uses the bitmap command to define the background for the display button. Refer to the following figure:


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\display_swap....
File
Edit
View
Format
Tools
Window
Help
INS
CAPS
NUM

```
Bitmap "c:\custom\displays\operating\common\bmp\_vlvl.bmp"
```

58 Characters Line: 1/1 Col: 54

9. The next two lines set up the conditions for the display swaps based on the value of the mouseclick variable. Refer to the following figure:

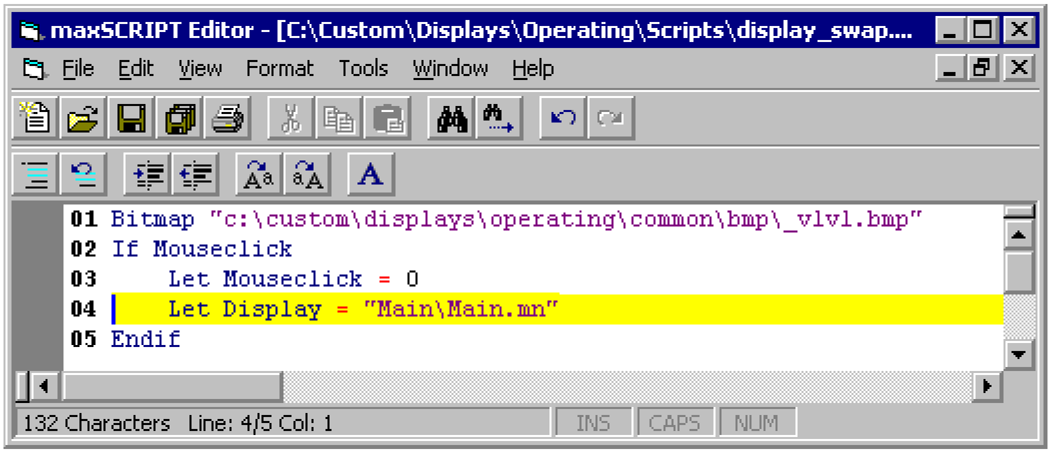

maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\display_swap....
File
Edit
View
Format
Tools
Window
Help
INS
CAPS
NUM

```
Bitmap "c:\custom\displays\operating\common\bmp\_vlvl.bmp"
If Mouseclick
    Let Mouseclick = 0
```

94 Characters Line: 2/3 Col: 6

10. The next line of script specifies the target display you wish to open with a mouse click. Enter the name of an existing maxVUE display, such as the displays you created in the previous exercises containing maxSCRIPT controls.

    Be sure the correct full file path and display name (including file extension) is specified. The file path must begin after C:\Custom\Displays\Operating. For example, *Main\Mainmenu2.MN*, *Boiler\Boiler.MN*, *Air\Air.MN*, etc. Refer to the following figure:


maxSCRIPT Editor - [C:\Custom\Displays\Operating\Scripts\display_swap....
File Edit View Format Tools Window Help
132 Characters Line: 4/5 Col: 1
INS CAPS NUM

```
Bitmap "c:\custom\displays\operating\common\bmp\_vlvl.bmp"
If Mouseclick
    Let Mouseclick = 0
    Let Display = "Main\Main.mn"
Endif
```

11. End the condition (endif) and save the script file using the maxSCRIPT Editor menu options.

Testing Exercise 3 in Runtime Mode

To see if the script you just created actually works, you must open it in maxVUE Runtime.

To test your script:

1. From the Windows NT desktop, double click on the maxVUE Runtime icon to open the application.

2. Click anywhere on the logo screen (but away from the maxVUE logo animation) to open the Main Menu display.

3. Click the navigation button you created to display swap to the target display you specified in your script, such as *mydisplay.mn* or whatever you chose to call it.

Studying maxSCRIPT Examples

After completing the proceeding exercises, you're probably ready to look at several real examples of maxSCRIPT scripts. The scripts in the following examples each incorporate some script coding ideas that you may want to borrow when you create your own scripts.

Example 1: Calculate an Average, Trigger an Alarm Indicator

The following script sets off an on-screen yellow alarm indicator when the state of two alarm attributes associated with two input values become true.

```
Bitmap
C:\custom\displays\operating\common\bmp\hide_bitmap.bmp"


Dim Input1 as intin(%param1%.out)
Dim Input2 as intin(%param2%.out)
Dim Output as intout(%param1%_avg.out)

Dim AnyAlarm1 as intin(%param1%.Anyalarm)
Dim AnyAlarm2 as intin(%param2%.Anyalarm)
Dim AnyAlarm as intout(%param1%_avg.Anyalarm)

let output = ( Input1 + Input2 ) / 2

if ( AnyAlarm1 = 1 ) or (AnyAlarm2 = 1 )
     let AnyAlarm = 1
Else
     let Anyalarm = 0
EndIf
```

The script begins with a bitmap command followed by a path name.

```
Bitmap
"C:\custom\displays\operating\common\bmp\hide_bitmap.bmp"
```

In this case, the bitmap is only use to hide the maxSCRIPT control on the display. The bitmap command calls out a background color that is identical to the screen display color, making the control blend in with its background and disappear from view. This is a convenient way to hide the control when it need not be visible, reducing screen clutter and confusion.

The next part of the script uses six dimension statements to read and write values to the Local Status Server in the Software Backplane.

```
Dim Input1 as intin(%param1%.out)
Dim Input2 as intin(%param2%.out)
Dim Output as intout(%param1%_avg.out)

Dim AnyAlarm1 as intin(%param1%.Anyalarm)
Dim AnyAlarm2 as intin(%param2%.Anyalarm)
Dim AnyAlarm as intout(%param1%_avg.Anyalarm)
```

The first two dimension statements are reading two input values from the parameterized tag names %param1%.out and %param2%.out.

The two inputs have two associated attributes for alarming called AnyAlarm1 and AnyAlarm2. The following dimension statements are reading these two values:

```
Dim AnyAlarm1 as intin(%param1%.Anyalarm)
Dim AnyAlarm2 as intin(%param2%.Anyalarm)
```

The script uses a *Let* statement to calculate an average for the two input values, Input1 and Input2. The following dimension statement is used to output the result of this calculation to a location in the Local Status Server called %param1%_avg.out.

```
Dim Output as intout(%param1%_avg.out)
```

For example, if the group parameter 1 is set to FIC101, a point named FIC101_avg with a member *.out* will be created. Another control, such as a bar control, can use this output via the identifier FIC101_avg.out.

Finally, the script uses an *If* expression to test for an alarm state. The expression states that if AnyAlarm1 or AnyAlarm2 enters a true state, then let an output value, created in maxSCRIPT, called AnyAlarm equal 1. Using a dimension statement, this value is then passed on to an output tag called %param1%_avg.Anyalarm. This then activities a screen alarm indicator. See the following dimension statement:

```
Dim AnyAlarm as intout(%param1%_avg.Anyalarm)
```

Example 2: Working with Text Strings

Use maxSCRIPT to call out text strings that are treated as constants. In this script example, a group of similar displays use the same display title, however, when one of these displays is accessed it needs to show a different title, "Oil Elevation A Front Sequence." This is accomplished with the following script.

```
Dimx TitleTxt1 as Strctlout(Title/TitleTxt1.text)
Dimx TitleTxt2 as Strctlout(Title/TitleTxt2.text)

Dim DP1 as Strin(%dp1%)

If dp1 = "OilSeqAF"
     Let  TitleTxt1  =  "Oil  Elevation  A  Front
Sequence"
     Let  TitleTxt2  =  "Oil  Elevation  A  Front
Sequence"
Endif
```

The script begins with two dimension statements that are writing to two tagname.attribute locations in the Local Status Server called Title/TitleTxt1.text and Title/TitleTxt1.text. Notice that the dimension statement includes an *x* (dimx). This means the script assumes it is accessing an existing variable, and does not need to create an object that is recreated and deleted each time maxVUE is opened and closed.

A third dimension statement is reading a text string from any maxVUE display file with the parameterized tag reference, dp1.

The script logic following the dimension statement states that if a display is accessed called "OilSeqAF," let the display title be "Oil Elevation A Front Sequence."

Example 3: Hiding a Button in a Pop-up Display

The following script was written for a pop-up display that is used for multiple screen applications. The display interacts with multiple maxSCRIPT controls at any given time, which dynamically change its appearance based on the state of current inputs.

```
'--------------------------------------------------
'This script hides the State2 command and feedback
'button when Numstate = 2.
'--------------------------------------------------

' Configure Runtime Bitmap to hide MAXscript.ocx
Bitmap "c:\custom\displays\operating\common\bmp\S-
popback-1.bmp"

Dim Numstate as Intin(%dp1%.Numstate)

Dim Height_rectangle as
dblctlout(tank/State2rectangle.height)
Dim Height_button as dblctlout(tank/state2.height)

Dim Width_rectangle as
dblctlout(tank/State2rectangle.width)
Dim Width_button as dblctlout(tank/state2.width)

If Numstate = 2
      let Width_rectangle = 0
      let Width_button = 0
      let height_rectangle = 0
      let height_button = 0
endif
```

The pop-up as created contains three buttons that may be assigned different functionality depending on the inputs the pop-up is currently referencing. In certain specific instances, however, only two buttons are relevant to the application that called the pop-up. The script shown here is designed to hide the third button when an input indicates only two buttons are needed.

The script begins with several comment lines set off by the initial single quote or apostrophe character. When the script is compiled, text strings that follow a single quote are ignored. The comments summarize the purpose of the script, which is to hide the Stated 2 command and feedback button when an input called Numstate equals 2.

The script continues with a bitmap command that displays a pop-up display file called *S- popback-1.bmp*.

The bitmap command is followed by five dimension statements. The first dimension statement is reading the attribute Numstate that is associated with a display identifier called dp1. See the following dimension statement:

```
Dim Numstate as Intin(%dp1%.Numstate)
```

The four other dimension statements define names to be used to write to the size attributes of the State2 button. Note that the size attributes include a rectangle that refers to the border surrounding the button.

Note: controls can be renamed using maxVUE Editor Layers button. Click the **Layers** button, locate and select the control of interest, right-click and select **Rename** from the menu and enter a new name. See discussion under "Interacting with Other maxVUE Controls," in Chapter 3.

The logic that follows the dimension statements states that if Numstate reads an input equaling 2 then write a zero to the size attributes of the button and surrounding rectangle. This effectively makes the button and button border disappear from the display when this script is in effect.

Note: when you select Test Mode in the maxVUE Editor, scripts can modify the parameters of other unsaved controls, such as width, height, x or y, or color. If you save the graphic after the script is run, these parameters changes will be remembered. Be sure to save prior to testing to prevent this.

By using script this way, you create a single display that may be tailored for multiple purposes, eliminating the need to create multiple displays designed to anticipate very specific uses.

Exercise 4: Using Flashing Bitmaps

In addition to the Bitmap command, maxSCRIPT includes the fbitmap command, which makes a bitmap image appear to flash repeatedly. The following script uses two flashing bitmaps to call attention to a device depicted on a display when it enters an alarm state.

```
Dim DevState as Intin(%param1%./#1.DevState)
Dim Ovrd1 as Intin(%param1%./#1.Ovrd1)
Dim Ovrd0 as Intin(%param1%./#1.Ovrd0)

Dimx SelPt as Strout(_sel_pt.Tagname)

if (Ovrd1 = 1) and (DevState = 2)

       Bitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
1.bmp"
       fBitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
1fl.bmp"

elseif (Ovrd0 = 1) and (DevState = 1)

       Bitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
2.bmp"
       fBitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
2fl.bmp"

elseif DevState = 2

       Bitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
1.bmp"
       fBitmap ""

elseif DevState = 1

       Bitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
2.bmp"
       fBitmap ""

else
       Bitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
5.bmp"
       fBitmap ""

endif

If mouseclick
       Let mouseclick = 0
       Let SelPt = "%param1%"
endif

If mousedclick
       Let mousedclick = 0
       let display = "popups\zpop-ct-dv2-o-s-c.mn dp1=%param1%"
endif
```

The script uses three dimension statements to define inputs for DevState, Ovrd1, and Ovrd0. The logic that follows the dimension statements reads the inputs for the above attribute values to determine which set of inputs makes any of a series of *if* expressions true.

For instance if Ovrd1 = 1 and DevState = 2, then the script displays the following two bitmaps using these path names:

```
"c:\custom\displays\operating\common\bmp\s-cb-a-
1.bmp"
"c:\custom\displays\operating\common\bmp\s-cb-a-
1fl.bmp"
```

The bitmap images are identical but use different shades of the color red. The first bitmap uses a medium red and the second bitmap uses a dark red. The first bitmap is called out using the Bitmap command, while the second bitmap is called out using the fBitmap command. To create the illusion of a single flashing image, the fBitmap command causes maxVUE in runtime mode to display both bitmaps alternating in rapid succession.

Note that the script is written to display only one of two flashing bitmaps when the proceeding *if* conditional logic is satisfied. The fBitmap command is used in three other places in the script followed by a null text string as in the following:

```
Bitmap "c:\custom\displays\operating\common\bmp\s-cb-a-
5.bmp"
fBitmap ""
```

When a script uses the fBitmap command, you must continue to use this command in other places that call for a bitmap even when a flashing bitmap is not desired. For instances where a flashing bitmap is not desired, use the *fBitmap ""* script construct to nullify or cancel a flashing bitmap command.

Chapter 3

maxSCRIPT Script Language Reference Guide

Overview

The maxSCRIPT OCX control uses a Basic-like script language, allowing you to create maxVUE display applications using a more free-form programming approach. The script language, developed by Metso Automation MAX Controls, uses elements, such as dimension statements, commands, variables, and logical expressions, providing you with a rich coding language. All the elements you may currently include in your script files are listed in this reference chapter. Each of the following sections describes the script element and its general syntax, and shows you how the element is typically incorporated in script using basic examples.

Using Dimension Statements

The following sections discuss the four basic uses of dimension statements in maxSCRIPT. Generally, dimension statements are used to define names that may be used to read and write data. Dimension statements may contain specific point identifiers, however, in most cases it is more advantages to use parameters instead. See "Using Parameterization," in chapter1.

Creating Local Variables

Local Variables can be created through dimension statements using the following syntax:

Dim <alias> as <type>

Where:

<alias> any string that uniquely identifies this value in the rest of the script
<type> is from

dbl – double variable
int – integer variable
str – string variable (String characters such as a description, or tagname)

Note when you first create them, double variable (0.0), integer variable (0) and string variable (“”) begin as null values. The following technique can be used to do something on the first pass only:

Example

```
Dim f as int
If f = 0
 Let f = 1
 …..do something
endif
```

Creating Objects in the Local Status Server

Use dimension statements to create objects in the Local Status Server, from which other controls can access information produced by maxSCRIPT. These objects are automatically created and deleted as maxVUE screens are called up. Use an Out type Dimension statement at the beginning of the script to declare these objects. Because there may be multiple instances of the same maxSCRIPT controls on a single graphic, names of output objects must be unique.

Syntax for Dim statements

Dim <alias> as <type>(<service>.<extended member>)

Where:

<alias> any string that uniquely identifies this value in the rest of the script
<service> any valid SBP service name
<extended member> any valid SBP extension
<type> consist of the following in and out types:

In Types:

xxxIn types refer to an already existing variable, which is to be used in a calculation.

dblIn – double in	Double precision, floating point integer; typically a variable used in a calculation.
intIn – integer in	Integer variable
strIn – string in	String characters such as a description or tagname

Out Types:

xxxOut types create (and delete when the screen closes) an LSS object for output (as well as input if necessary) so that other controls can access the calculated results. A <service> is created for the purpose of housing outputs in the Local Status Server. The same service can (and should) be used for multiple members. The members can be of all different types.

DblOut – double out	Double precision, floating point integer; an output to be created so that controls can access calculated results
IntOut – integer out	Integer variable
StrOut – string out	String characters such as a description or tagname

Examples:

```
dim xyz as dblOut(b.condou)
dim i12 as intOut(b.conint)
dim txt as strOut(b.contxt)

dim asaw as dblIn(saw.ao)
dim lss_service_count as intIn(_lss.numsvc)
dim now as strIn(_lss.time)
```

Caution: The statement

```
Dim A as strout(_sel_pt.tagname)
```

has the unfortunate effect of deleting the selected point object when the script is terminated. It is up to the script writer not to use existing local status variable names in an xxxOut dimension. Instead, use <Scriptname>.out or something similar.

Performing Demand Writes Using Command Statements

In many situations, maxSCRIPT will need to write the same value multiple times during a process. For these instances use a dimension statement that incorporates a command, a type of transaction supported by DPU4E.

A command is an action that takes place upon a write from the software backplane. Although commands are not readable, you might write the same value each time (0, for example).

Syntax for Dim statements:

Dim <alias> as <type>(<service>.<extended member>)

Where:

<alias> any string that uniquely identifies this value in the rest of the script
<service> any valid SBP service name
<extended member> any valid SBP extension
<type> consist of the following out type:

IntCmdOut	Integer variable output
sCmdOut	String variable output

Example

```
dim gotoman as intcmdout(%param1%.opcmdman)
if mouseclick
     let gotoman = 0
endif
```

Without the use of the command statement, maxSCRIPT only writes the value once, assuming nothing has changed since the last write. The command statement, in effect, forces a demand write. maxSCRIPT will perform the write regardless of what was written to this command the last time, and it will not subscribe to the command to update current values.

Note that the maxSCRIPT variable Display (See Table 1-1 above) uses the sCmdOut command statement on behalf of the user.

Note that normal outputs DO subscribe to values and can be used as inputs.

Example:

```
dim a as intout(test.out)
if a = 5
     let a = 0
endif
```

Note that *a* can be read and written by the script.

Performing External Writes

When accessing an existing object (such as _sel_pt) as an output, write the dimension statement as *dimx*.

```
dimx s as strout(_sel_pt.tagname)
```

When out types are used with the dimx command, an existing variable is assumed (and not created or deleted).

Use *dimx* to write to existing software backplane service members, such as DPU point (service) attributes (members) and LSS service members.

The function of the *x* is to prevent script from creating an _lss variable of the name (<external name>.<external attribute>), which is deleted when the script is terminated.

Dimx is only functional for:

```
dimx <name> intout(<external name>.<external
attribute>)
dimx <name> dblout(<external name>.<external
attribute>)
```

```
dimx <name> strout(<external name>.<external
attribute>)
```

Should you write a dimx statement for xxxIn types, the script will perform like a dim statement without the x. The *x* is ignored for inputs.

```
dimx <name> intin(<external name>.<external
attribute>)
dimx <name> dblin(<external name>.<external
attribute>)
dimx <name> strin(<external name>.<external
attribute>)
```

Caution: because there is potential for security bypass or inadvertent operation, you must exercise great caution when you use the dimx script instruction.

Syntax

Dimx <alias> as <type>(<service>.<extended member>)

Where:

<alias> any string that uniquely identifies this value in the rest of the script
<service> any valid SBP service name
<extended member> any valid SBP extension
<type> is intout, dblout or strout

Example

```
Dimx selpt as strout(_sel_pt.tagname)
let selpt = “fic101”
```

Subscribing to Keyboard Groups Using _Keyboard.respool

Use a dimension statement containing `Dimx` to access *_keyboard*, a Local Status Server (LSS) member service. This object is used in script to permit applications, such as displays containing buttons, to respond to keystrokes from the operator keyboard.

The service, _keyboard, is used to buffer keystrokes from the operator keyboard. A subscription to the keyboard takes the form:

Subscribe _KEYBOARD.BUFFER.<MASK>

Where <MASK> indicates the desired key clusters, represented by a single letter:

M - ode
D - isplay
P- an
C - ursor

Z - oom
U - ser
L -ogic
S - etpoint
O - utput
A –alarm

For example:

Subscribe _KEYBOARD.BUFFER.MDP to subscribe to mode, display and pan key clusters.

An application can subscribe to a group of keys. Only the last subscriber to a group of keys gets notified that the key has been pressed. The respool function of _keyboard:

_keyboard.respool

allows an application to throw back any keys that it is not interested in.

In dimension statements containing respool, use dimx as part of the statement since the script is not creating the _keyboard.respool variable. It is built into lss.

Use intcmdout in dimension statements performing writes to _keyboard.respool. This is a dim type that does not subscribe to _keyboard.respool. Note that you cannot use respool in a read mode – if (respool = 4) doesn’t make sense.

Intcmdout has a second feature. Since it does not subscribe to a value, it will write the same value over and over again. The normal dimx output point will not write unless a value is different from the last value received from a point.

The following is an example maxSCRIPT that intercepts a group of keys:

```
Rem subscribe to the user keys ‘u’
Dim key as intin(_keyboard.buffer.u)

If (key = 115)
     Rem do f4 special processing
Endif
```

The dimension statement in the previous example is subscribing to the user function keys:

```
_keyboard.buffer.u
```

Unfortunately, this fragment has the side effect of intercepting all of the User function keys. If this script is not capable of handling all of the keys it has to put them back for the next subscriber. This is resolved in the next example containing the `Dimx respool` statement.

```
Rem subscribe to the user keys ‘u’
Dim key   as intin(_keyboard.buffer.u)
Dimx respool as intcmdout(_keyboard.respool)

If (key = 115)
     Rem do f4 special processing
Else
     Let respool = key
Endif
```

Finding Keycodes

Notice that the script in the preceding example references a specific function key by its number code, 115. The following script can be used to find keycodes:

```
Dim key as intin(_keyboard.buffer.ncpdzumlsoai)
Dim out as intout(test.out)

Let out = key
```

Put this script on a graphic, with a single point displaying the integer at test.out, and you will see keys as they are pressed. Notice that a key is always followed by a zero around one second after the key has been pressed.

Interacting with Other maxVUE Controls

You may use dimension statements that allow a maxSCRIPT control to interact with other maxVUE controls, such as the Bar Control, List Control, and so forth.

Your script should include one of the following out types

Dblctlout; double integer out to control	Double precision, floating point integer an output to be created so that controls can access calculated results.
Intctlout: Integer out to control	Integer value
Strctlout; string value out to control	String characters such as a description, or tagname

To interact with other controls on a given maxVUE display, use the following syntax:

Dim <name> as <xxx>ctlout(<object name>.<parameter>)

Where:

<name> is the alias for this parameter within maxSCRIPT

<xxx> is replaced by Dbl, Int or Str; note that the out types include the characters *ctl*, indicating the script will interact with other controls. For example, Dbl***ctl***out.
<object name> is the name of the object as seen and set in the Layers button of the maxVUE Editor
<parameter> is from the list:

x - the x position
y - the y position
width - the width of the object
height - the height of the object
color - the color of the object
linecolor - the linecolor of the object

Example:

```
Dim x as intctlout(object.x)
Dim y as intctlout(object.y)
Let y = 10
```

Note: when you select Test Mode in the maxVUE Editor, scripts can modify the parameters of other unsaved controls, such as width, height, x or y, or color. Be sure to save any unsaved controls prior to testing to prevent this.

Color is best represented by a *dbl* value. The *dbl* value should be calculated as:

```
Dim objcolor as dblctlout(object.color)
Dim color as dbl
Dim red as int
Dim green as int
Dim blue as int
Let red = …                ;number from 0 to 255
Let green = …        ;number from 0 to 255
…

Let color = RGB (red, green, blue)

Let objcolor = color
```

When creating a maxVUE group with a script interacting with another control using this syntax:

intctlout(<controlname>.<attribute>)
dblctlout(<controlname>.<attribute>)
strctlout(<controlname>.<attribute>)

you need to know the name of the target control, which may be any available maxVUE control, such as a Button Control, Box, and so forth).

To give a control a name:

1. Click the Layers Button to display the Edit Layers and Control Order pop-up of the selected file.

2. When you access the Edit Layers and Control Order pop-up, display objects appear in Folders, organized in a tree structure.

3. To change the default name to any name of your choosing, select the object and right-click to bring up a pop-up menu. Click **Rename** from the menu and enter a new name.

All Group Folders (except the root) may be renamed. The individual drawing items within a Folder may also be rename. It IS OK that the names are not unique when a group is copied and pasted. This function will first search for a control within the group with <controlname>. It will then look at the next level of grouping and so forth.

Using maxSCRIPT Commands

maxSCRIPT uses the following commands:

- Let
- If, else, etc
- Bitmap/fbitmap
- Play
- ToolTip
- Run

Using Let X = Y Expressions

Expressions beginning with *Let* use the following syntax:

Let <service>.<extended member> = <expression>

Where:

<expression> is any valid arithmetic or logical expression containing the following:

<service>.<extended member>	Previously declared as a dblIn, intIn, or strIn
<constant>	Numerical or string constants
+ - / *	Arithmetic operators. Note + also stands for string concatenation
( )	Parenthesis
and or not bitand bitor	Logical operators for logic expression
> < =	Relational operators and combinations

Example:

```
let xyz = 4.5432 * asaw
let txt = "testing at " + now
```

Bitand and bitor are used to AND or OR two operators together to produce either a zero or non-zero result. Typically, one of the operators will be an integer constant whose decimal equivalent is one or more bits set, while the other is a multi-bit integer which is to be tested for the state of a bit buried within the integer.

Example of use of bitand (example taken from the popup used for the Digital Input Buffer); the goal of the code is to animate individual bits of a word; thus, through parameters, this code is called 16 times in the Detail Popup.

```
Bitmap = "C:\mcs\displays\common\bmp\backgray.bmp"

Dim bn as Int
Dim bit as Int
Dim word as intin(%param1%.%param3%)
Dim start as intin(%param1%.BIT_NUM)
Dim bits as intin(%param1%.FIELD_SZ)
Let bn = %param2%

Dim BitOut as Intout(%param1%%Param2%_Bit.Out)

Let bit = 1 bitand word

If (bn < start) or (bn > ((start + bits) - 1))
      If bit
            Let BitOut = 3
      Else
            Let BitOut = 2
      Endif
Else
      If bit
            Let BitOut = 1
      Else
            Let BitOut = 0
      Endif
Endif
```

Using If, Else, Else If, EndIf Expressions

Expressions using *If*, *Else* statements use the following syntax:

```
If <expression>
Else
ElseIf <expression>          (Note one word)
Endif
```

Where:

<expression> is any numerical expression that can be checked against zero.

Example

```
if asaw > 50.
 let txt = "saw gt 50"
else
 let txt= "saw lt 50"
endif
if (xyz > 3.0) and (txt = “on”)
```

Using the Bitmap Command

The maxSCRIPT control lets you display bitmaps using the Bitmap command. Access the control property page called Default to define a file, such as a bitmap that you want to appear in the maxSCRIPT window by default to replace the standard maxSCRIPT icon.

When you create the control in the maxVUE Editor, you may scale the window to any size including full screen. The bitmap assumes the size of the control window.

You can control how and when a bitmap or multiple bitmaps appear in run mode using the bitmap instruction by itself or with additional script logic:

Bitmap “<expression>”

Where

<expression> is the file path and name to the desired bitmap.

```
Dim f as int
If f = 0
 Let f = 1
 Bitmap “c:\winnt\winnt.bmp”
Endif
```

You can calculate a bitmap via an expression:

```
Dim str as strin(bmp.name)
Bitmap   “c:\custom\displays\operating\bitmaps\”   +
str
```

Using Flashing Bitmap Command

To cause a flashing bitmap, use the fbitmap option. This will cause a continuous switch between the normal bitmap (the last bitmap specified) and a second bitmap on a half second basis.

```
bitmap "C:\winnt\winnt.bmp"
fbitmap "C:\winnt\winnt256.bmp"
```

You reset the flash by setting fbitmap to ""

```
fbitmap ""
```

You can reset the name of the normal bitmap and flashing bitmap at anytime.

Using the Play Command

Use the play command to call out a sound file. In the following example the wave file called out by the play command produces a sound.

```
play "c:\winnt\media\start.wav"
```

Using the ToolTip Command

Use the ToolTip command to add a ToolTip window to a maxSCRIPT control. The ToolTip may contain a text message that describes the control to an end user. The ToolTip appears when a user rests a mouse cursor over the control. To activate the feature and copy a text message to the ToolTip window, use this syntax:

```
Tooltip "<Text string>"
```

To create a ToolTip that displays the last error message for the maxSCRIPT control, combine the ToolTip command with the Error variable. See "Using the Error Variable," in the following section.

Using the Run Command

Use the run command to run a program. In the following example the Report Control Program is run when the maxSCRIPT control is clicked.

```
If MouseClick
 Let MouseClick = 0
 Run "c:\Mcs\Report\ReportControlPanel.exe,Reports Control Panel"
Endif
```

The Run Command is followed by a comma-delimited string of two parts. The first part is the full pathname of the program to be run. The second part is the program’s associated Window Title. The two parts are separated by a comma. When attempting to run the program, maxSCRIPT first checks to see if it is already running by matching the Window Title in the Run Command with Window Titles of running processes. If not found, it will run the program. If it finds it, the program will be brought to the forefront of the desktop. For this reason, it is extremely important that the correct Window Title be used with the Run Command. To determine the correct Window Title, double click on the program and then wave your cursor over its Icon on the Task Bar.

The Window Title may include the wild card character (*) to match multiple characters, or the position wild card character (?) to match any character in a position:

```
If MouseClick
 Let MouseClick = 0
 Run "C:\Winnt\System32\mspaint.exe,*paint"
Endif
```

Using Special Variables

maxSCRIPT uses three special variables:

- Display
- Mouseclick, Mousedclick, Mouserclick
- Error

Using the Display Variable

Use the Display variable to request a popup or display change:

```
Let display = “main\main.mn”
```

Using the Mouse Clicks Variable

Use the following mouseclick variables to create an object that responds to mouse clicks:

Mouseclick: processes left click over this control

Mouserclick: processes right click over this control

Mousedclick: Processes double click over this control

```
     If mouseclick = 1
          Let mouseclick = 0    ;need to reset it
          ......
     Endif
```

Using the Error Variable

The variable *error* contains the last script error. For example,

```
tooltip error
```

will copy the last error to ToolTip. Error can be set by applications software:

```
let error = "broken"
or
let error = ""
```

Error will remain set until a new error occurs.

Using Variables Mouserow/ Mousecol to Locate Mouse Cursor

Use the variables Mouserow/Mousecol in a maxSCRIPT to determine the location of the mouse cursor.

Note the units of row and col are percent of the graphic and can be placed into x and y directly.

See the following example.

Mouse row and column are only calibrated in runtime.

Example:

```
rem mouse row and col demonstration script
dim drow as dblout(test.drow)
dim dcol as dblout(test.dcol)
dim time as strin(_lss.time)
dim s  as str
rem circ is a circle object that has been rename
circle
rem it is an example of moving an object to follow
the cursor
dim row as dblctlout(circ.x)
dim col as dblctlout(circ.y)
if mouseclick
 let s = "The time is " + time
 let s = s + ",Clock"
 let mouseclick = 0
endif

let s = "col=" + mousecol + " row=" + mouserow
let row = mouserow
let col = mousecol
let drow = mouserow
let dcol = mousecol
tooltip s
```

Using Special Functions

maxSCRIPT uses the following special functions:

- ABS; Absolute Value
- ASC or CHR; convert number to ASCII character
- ASK; present the user with a message popup
- AVI; specify and display frames
- EXP; expand parameters in real time

- NOT; Logical Negation
- OBJMAP; perform lookup in typemap database
- P_CONTROL; perform lookup of the name of a control popup display
- P_DETAIL; perform lookup of the name of a detail popup display
- QUA; check variable quality
- RGB; calculate a color
- SendMsg; send a message to another window
- SVC; select a service
- TRANS; enable dual language translator
- VAL; convert ASCII string to double

Refer to the following for examples of each function.

Using the Absolute Value Function

ABS – Absolute Value

Use the following syntax:

ABS(Expression)

Example:

```
Dim dev as dbl
Dim pv as dblin(tagname.pv)
Dim sp as dblin(tagname.lsp)
Let dev = abs(pv-sp)
```

Using the ASCII Character Function

Use the asc function to convert a number into an ASCII character. “chr” is accepted by maxSCRIPT as a synonym for “asc”.

Example:

```
dim i as intout(script.int)
dim s as string
let s = "The " + asc(i) + "time"
```

Note there are no errors. Asc will truncate all numbers from 0 to 255 without raising an error.

Example:

```
asc(256 + 35)= '#'
```

Using the ASK Function

The ASK function places a message popup in front of the operator so that the operator can make a decision. The operator must always answer the message popup.

ASK supports three different popup types:

Yesno: the operator will be asked yes or no to answer a question. The result will be 1 if the operator clicks on Yes, or 0 if the operator clicks on No.

Input: the operator will be asked to enter a string variable, which will then be placed in the result.

Alert: the operator will merely have to acknowledge the popup; no result will be returned.

The result, if any, will be returned in the special variable *askvalue*.

The ASK function is invoked in the following manner:

Ask “<text message>|<popup caption>|<popup type>”

Where

<text message> would be replaced by the actual text message to be conveyed to the operator, such as `Enter Tag:`

<popup caption> would be replaced by the title of the message box, such as `Add Trend Point`

<popup type> would be either `yesno` or `input` or `alert`

Here is an example, taken from ControlPopup.mxs, showing one use of the ASK function:

```
dim s as strin(_sel_pt.tagname)
dim d as str
if mouseclick
  let mouseclick = 0
  if s <> ""
    let d = p_control(s)
    if d <> ""
      let d = d + " dp1=" + s
      let display = d
    else
      ask "Selected Point Not Recognized|No Control popup|alert"
    endif
  endif
endif
tooltip "Click to call up a Point Control popup for the selected
point.  Note - You must Select a point first."
```

In this example, the code generates an alert-type message box, with text message Selected Point Not Recognized, and with title No Control popup, if the p_control function does not return a display name.

Here is another example. Using this fragment, an operator can, using ASK, call up a trend, and then enter a tagname and an attribute to be trended. An alternative method to calling up the Trend Attributes popup.

```
Ask “Enter Tag:|Add Trend Point|input”
if askvalue <> “”
  Let Pen1t = askvalue
  Ask “Enter Attribute:|Add Trend Attribute|input”
  Let Pen1a = askvalue
  Let PointId = Pen1t + “.” + Pen1a + “.sample”
endif
```

Using the AVI Function

Use AVI function to select a specific frame and display it. It is not intended to play movies; use the time-state control for that. A typical AVI application would be an oddly shaped tank being filled. You might create 100 frames showing various states of the tank. You can then select one of the frames. An alternate bitmap approach, using 100 if statements, would be too cumbersome.

Before using an AVI it must be loaded. In the following example, this script is used to load an AVI called clock:

```
dim first as int
if not first
let first = 1
avild "clock.avi"
endif
```

Use the following syntax to display a frame:

```
avi <frame>
```

Where <frame> is an integer expression representing the frame number.

Frame Rate

The frame rate for new frames is 1/2 second. For example, the following code will display the frames 1,2,3,5,7 in sequence at a rate of 1/2 second.

```
if gonow
let gonow = 0
avi 1
avi 2
avi 3
avi 5
avi 7
endif
```

Using the Expand Function

The Expand function performs a real-time expansion of parameters.

Example:

```
dim sin as strout(junk.in)
dim sout as strout(junk.out)
let sout = exp(sin)
```

Put a list control with two entries (writes on junk.in) and you can type in a string (%param1%.%dp1%) and see what it gets expanded to.

Expansion can be used to perform an ad-hoc read via maxSCRIPT. In the following, a tag and attribute are computed depending on an input variable. The read is then executed via the expand function.

```
Dim s as str
dim d as dbl
Dim i as intin(test.in)
let s = "%%"
if i = 0
let s = s + "fic101"
elseif i = 1
let s = s + "lic101"
endif
let s = s + ".pv%%"
let d = exp(s)
```

Note the double parenthesis characters (%%) are used because lines of script are pre-expanded prior to compilation. %% expands to a %.

Using the Logical Negation Function

NOT – Logical Negation – logical signals are either zero (0) – false or non-zero – true. The *not* operator will convert a zero to a 1, and a non-zero to a 0.

Use the following syntax:

NOT (Expression)

Example:

```
if not (a = b)
```

Using the Objmap Function to Access the TypeMap

Use the Objmap function to look up the mapping of an object type (such as PID, SUMMER etc.) to find its appropriate point detail and point data popup.

The function references a database file listing all known object types and their associated point detail and point data pop-up displays by specific file name. All objects in systems from MAX1 through maxDNA have an object type such as PID, SUMMER etc. This file is located in:

C:\custom\displays\operating\database\typemap.mdb

The database file table consists of the following columns, which can be filled in:

PointDataPopup
PointDetailPopup
PointControlPopup
UserDefined1
UserDefined2
UserDefined3
UserDefined4
Help

Note: The only ones in service for maxDNA are PointDataPopup and PointDetailPopup.

The objmap function uses the following syntax:

Objmap(<object map string>)

Where

<object map string> = <objtype> | <column name>
<object> = <pointid>.objtype
<column name> is from the above list of columns.

Example

Following is a fragment of a program to call up the PointDataPopup of the selected point.

```
dim obj as strin(_sel_pt.route.objtype)
dim s  as str
if mouseclick
 let mouseclick = 0
 let s  = obj + "|PointDataPopup"
 display objmap(s)
endif
```

Note that the temporary string ‘s’ is used to compute the string consisting of the object type and the column to be accessed separated by the vertical bar.

Objmap will return an error message in the string should an object type not be found, or the column name not be found. In addition, you can test the quality of an Objmap function to determine if the lookup is successful:

Example

```
If qua(objmap(“%_sel_pt.forward.objtype%” +
“|PointDataPopup”)
        …
endif
```

Note that the above makes use of parameter substitution to read “_sel_pt.forward.objtype”.

Using the P_CONTROL and P_DETAIL Functions

These functions return the name of a point control or point detail popup display, given a point tagname as input.

For DBM-based points, the TypeMap.mdb database file is searched for the name of a popup, which will typically come from either C:\Mcs\Displays\Mn\PointData folders, or C:\Mcs\Displays\Mn\PointDetail folders (but, of course, it is possible for you to create a modified set of displays in the Custom path, after modifying the TypeMap.mdb file).

For DPU4E-based points, the Point Control popups will typically come from C:\Mcs\Displays\Popups\dpmsxxx folders (where xxxx is the name of an atomic block), while the Point Detail popups will typically be the PtDetailsTabular.mn display (looks like the Point Browser), although you could create custom Control or Detail popups which would be stored in the C:\Custom\Displays\Operating\Popups path.

For DPU4E-based atomic blocks, the display names are always going to be stored in a folder C:\Custom\Displays\Operating\Popups\<objname>, with file name either <objname>-control.mn or <objname>-detail.mn. For example:

The MAMC-supplied Point Control popup for the Atag atomic block is:

C:\Mcs\Displays\Mn\Popups\dpmsATAG\dpmsATAG-control.mn.

If you had created a custom version, it would be:

C:\Custom\Displays\Operating\Popups\dpmsATAG\dpmsATAG-control.mn.

These path and file names are the template for what you would use if you had created your own DPU4E Custom Function Blocks, and wanted to provide them with either a Control or Detail popup.

Here is an example of the use of the P_CONTROL function:

```
dim s as strin(_sel_pt.tagname)
dim d as str

if mouseclick
  let mouseclick = 0
  if s <> ""
    let d = p_control(s)
    if d <> ""
      let d = d + " dp1=" + s
      let display = d
    else
      ask "Selected Point Not Recognized|No Control popup|alert"
    endif
  endif
endif
tooltip "Click to call up a Point Control popup for the selected
point.  Note - You must Select a point first."
```

The variable s contains the tagname of the selected point. The variable d contains the path and name of the popup found by p_control.

Using the QUA Function

maxSCRIPT (Release 1.6.5 or later) includes enhancements to ensure data quality using the Qua function. maxSCRIPT versions released before Release 1.6.5 may execute with some unexpected consequences.

Example:

```
dim s1 as strin(input1.tagname)
dim s2 as strin(input2.tagname)
dim same as int
if s1 = s2
let same = 1
endif
```

In maxSCRIPT releases prior to Release 1.6.5, s1 and s2 are initialized to blank, and subscriptions are issued to update these variables as they change. The if statement *will most often* be true, (blank = blank), the very first time it executes.

In post Release 1.6.5 maxSCRIPT, s1 and s2 are initialized to Bogus (bad). The if statement first makes the calculation:

```
s1 = s2
```

If any variable in a calculation is bad (timed out, tag not found, data not available yet) the result is zero or false. So the above program will work. There are still situations where it would not as in the following example:

```
if (s1 <> s2)
same = 0
else
same = 1
endif
```

To be absolutely sure the data is good incorporate the Qua function in scripts as shown in the following example:

```
dim s1 as strin(input1.tagname)
dim s2 as strin(input2.tagname)
dim same as int
if qua(s1 = s2)
if s1 = s2
let same = 1
endif
endif
```

Note: the Qua function is performed on the result of the calculation s1 = s2. The result is thrown away.

Caution: some scripts that used to operate under Release 1.6.4 may operate differently under Release 1.6.5 due to the introduction of quality. In the first example, your program may depend on falling through the comparison (even though it is an error to do so)!

Using the RGB Function

The RGB command calculates a double precision integer representing a color.

Use the following syntax:

RGB ("r,g,b")

Example:

```
dim c as dblctlout(box.color)
let c = rgb "255,0,128"
```

The following example sets a full red, no green and half-blue intensity. RGB returns a double and can be stored in any *dbl* variable.

Example:

```
dim c as dblctlout(box.color)
dim d as dbl
let d = rgb "255,0,128"
let c = d
```

You may use the RGB function with the color and linecolor attributes of a control using this syntax:

```
dim co as dblctlout(<object name>.color)
Let co = RGB (128, 25, 174)
```

Using the SendMsg Function

This is a system function that can be used to broadcast a message to another window. You should not use this function if you don’t know what that means. It is primarily used to popup the point picker. The following example will popup the point picker on a mouseclick. The function was generalized as it may have other applications.

The following script should popup the point picker on a mouse click:

```
If mouseclick
  Let mouseclick = 0
  SendMsg “Point Picker WakeUp”
Endif
```

Using the Service Function

Use the SVC function to pick a service out of a full point identifier

Use the following syntax:

SVC ("service.member")

Example:

```
let str = svc(“fic101.pv”) ; str gets “fic101”
```

Using the TRANS Function

Use the TRANS function to run a string through the dual language translator. Note you will need to be careful with fonts in displaying junk.out.

```
dim sin as strout(junk.in)
dim sout as strout(junk.out)
let sout = trans(sin)
```

Using the Value Function

Use the val function to convert an ASCII string into a double precision integer.

Example:

```
dim i as int
dim s as strout(script.txt)
let i = val(s) 'convert an input string into a number
```

Note there are no errors reported. Val will convert as many characters as are appropriate and stop without error in any case.

Example:

```
val("123.4hi there") = 123.4
val("hi there123.4") = 0.0
```

Creating Unique Output Names

When creating external variables with an output dimension, use a method to create a unique output name to prevent interference among different instances of maxSCRIPT. The following will result in a unique name for the service to contain output variables:

```
Dim out as dblOut(%param1%_scr.out)
```

Note that the “_scr” addition to one of the driving tagnames creates a somewhat unique place in which to store the output of the script.

Changing maxSCRIPT Execution Time

maxSCRIPT subscribes to its data and will ONLY execute lines when there is a data update (or subscription timeout) to any of the values that are dimensioned as inputs or outputs. If you wish a more regular execution for some reason, simply use a dimension statement for a string input pointing to _lss.time that will update each second.

```
Dim dummy as strin(_lss.time)
Dim intout as intout(test.int)
Dim I as int
Let I = I + 1
Let out = I
```

Without the declaration of dummy, the above script will only execute when the *out* subscription times out..

Annotating Script with Comment Text

To annotate maxSCRIPT with free form comments, notes, special instructions, and so forth, begin each comment line with any of the following:

Rem
; Semicolon character
' apostrophe character

When maxSCRIPT is executed, any text beginning with the above characters is ignored.