



JScript Reference

Summary

Technical Reference

TR0122 (v1.0) December 01, 2004

This reference manual describes the JScript scripting language used in DXP.

Exploring the JScript language

The following topics are covered in this reference:

- Introduction
- JScript source files
- Creating new scripts
- Adding scripts to a project
- Executing a script in DXP
- Assigning a script to a menu, key or toolbar
- About JScript examples
- Writing JScript Scripts
- JScript keywords
- JScript Statements
- JScript Functions
- Forms and Components.

Introduction

This Reference details each of the JScript Scripting statements, functions and extensions that are supported in the scripting system. The Javascript Scripting or JScript for short also can deal with DXP Object Models and Visual Components. It is assumed that you are familiar with basic programming concepts and as well as the basic operation of your Design Explorer-based software.

The scripting system supports the JScript language (along with other scripting languages) which is derived from Microsoft Active Scripting system. All scripting languages supported in DXP are typeless or untyped which means you cannot define records or classes and pass pointers as parameters to functions for example.

JScript Example

```
function DisplayDate() {  
    var d, s = "Today's date is: "; //Declare variables.  
    d = new Date();                //Create Date object.  
    s += (d.getMonth() + 1) + "/"; //Get month  
    s += d.getDate() + "/";        //Get day  
    s += d.getFullYear();          //Get year.  
    showMessage(s);                //Show date.  
}
```

For detailed information on JScript and its keywords, operators and statements, please refer to Microsoft Developers Network website, <http://msdn.microsoft.com/library/en-us/script56/html/js56jsoriJScript.asp>

DXP and Borland Delphi Run Time Libraries

The Scripting system also supports a subset of Borland Delphi Run Time Library (RTL) and a subset of DXP RTL which is covered in the DXP RTL Reference in the Scripting Online Help in DXP.

There are several Object Models in DXP; for example you can use the PCB Object Model in your VB Scripts to deal with PCB objects on a PCB document, WorkspaceManager Object Model to work with Projects and their documents and extract netlist data for example.

The Scripting Online Reference Help contains information on interfaces with respect to DXP Object Models, components, global routines, types, and variables that make up this scripting language. There are several Object Models in DXP; for example you can use the PCB Object Model in your JScripts to deal with PCB objects on a PCB document, WorkspaceManager Object Model to work with Projects and their documents and extract netlist data for example.

Server Processes

A script can execute server processes and thus server processes and parameters are covered in the Server Process Reference.

JScript source files

You open a script project in DXP and you can edit the contents of a script inside the DXP. A script project is organized to store script documents (script units and script forms). You can execute the script from a menu item, toolbar button or from the *Run Script* dialog from the DXP's system menu.

PRJSCR, VBS and DFM files

The scripts are organized into projects with a PRJSCR extension. Each project consists of files with a js extension. Files can be either script units or script forms (each form has a script file with js extension and a corresponding form with a dfm extension). A script form is a graphical window that hosts different controls that run on top of DXP.

However it is possible to attach scripts to different projects and it is highly recommended to organize scripts into different projects to manage the number of scripts and their procedures / functions.

Scripts (script units and script forms) consist of functions/procedures that you can call within DXP.

Creating new scripts

You can add existing or new scripts into the specified project in the **Projects** panel in DXP. There are two types of scripts : Script Units and Script Forms. With a project open in DXP, right click on a project in the **Projects** panel, and a pop up menu appears, click on the **Add New to Project** item, and choose **VB Script Unit**. A new script appears.

A script can have at least one routine which defines the main program code. You can, however, define other routines and functions that can be called by your code. The functions and subroutines are defined within a **Function End Function** or **Sub End Sub** statement block.

Note that it is possible to have no routines within a script but at least it is necessary to have a .

Example of a Subroutine

```
Sub Log(Description, Data)
    Call ReportFile.Write(Description)
    Call ReportFile.WriteLine(Data)
End Sub
```

Example of a subroutine less script.

```
' script here with no function/sub routine
A = 50
A = A + 1
ShowMessage(IntToStr(A))
```

Adding scripts to a project

You can add existing scripts to a specified project in the **Projects** panel in DXP. With a project open in DXP, right click on this project in the **Projects** panel, and a pop up menu appears, click on the **Add Existing to Project...** item.

A *Choose Documents To Add to Project* dialog appears. You can multi-select as many scripts you want to add into the specified project.

Executing a JScript in DXP

In Text Editor workspace

You can configure the **Run** command when you are in the text editor to point to a script and execute it. Every time you click on the **Run** icon from the Text Editor menu or press **F5**, the scripting system executes the script pointed to by the **Set Project Startup Procedure** item. You can change the start

up procedure by clicking on the **Set Project Start Up Procedure** item in the **Run** menu which invokes the *Select Item to Run* dialog. You can then select which procedure of a script to be set.

Executing a script on a design document

To execute a script in DXP, there are two methods and there are two different DXP dialogs for each method. These methods are necessary if you wish to run a script on a server specific document such as PCB or Schematic documents.

1. Using the Select Item To Run dialog to execute a script

Click on the **Run Script** from the DXP system menu and the *Select Item to Run* dialog appears with a list of procedures (those parameter-less procedures/functions only appear) within each script in a opened project in DXP.

Note that you can also click on a script unit filename within this *Select Item to Run* dialog and the functionless/procedureless **Begin End.** block within the script gets executed. See code example here

Script unit

```
// script here with no function/procedure
A = 50;
A = A + 1;
ShowMessage (IntToStr (A) )
```

Now, only parameter-less functions and procedures for each script of an opened project only appear on the *Select Item to Run* dialog. It is a good idea for script writers to write the functions in scripts so that they will appear in this dialog and the other functions with parameters not to appear in this same dialog.

When you are working in a different editor such as PCB editor, you can assign the script to a process launcher and use it to run a specified script easily. See the *Assigning a script to a process launcher*.

You can add a list of installed script projects so that, every time you invoke the *Select item to Run* dialog, the installed script projects will appear along with other script projects currently open in the **Projects** panel. Invoke **Scripting System Settings** item from **Tools » Editor Preferences** menu in the TextEditor workspace and then drill down to the DXP System, Scripting System, and the *Scripting System page* appears.

2. Using the Run Process dialog to execute a script

Invoke the *Run Process* dialog from DXP's System menu and execute the **ScriptingSystem:RunScript** process in the Process: field and specify the script parameters, the **ProjectName** parameter which is the path to the project name and the **ProcName** parameter to execute the specified procedure from a specified script in the Parameters: field.

You need the following parameters for the **ScriptingSystem:RunScript** process to execute a specified script.

Process:

ScriptingSystem:RunScript

Parameters:

ProjectName (string)

ProcName (string)

Example

Process: ScriptingSystem:RunScript

Parameters : ProjectName = C:\Program Files\Altium2004\Examples\Scripts\VB Scripts\HelloWorld.PrjScr | ProcName = HelloWorld>HelloWorld.

To run a script repeatedly in the text editor in DXP, assign the script to the **Set Project Startup Procedure** item from the **Run** menu of the Text editor server. you can then click on the Run button. or press F5 to execute this script. To run a different script, you will need to re-invoke the **Set Project Startup Procedure** from the Run menu and assign a new script to it.

You can click on the **Run Script** item from the DXP system menu and the *Select Item to Run* dialog appears with a list of procedures (those parameterless procedures/functions only appear) within each script in a project. This may be needed if you wish to run a script on a specific document type such as PCB or Schematic documents.

You can also use the *Run Process* dialog and specify the scriptingssystem server process and specify the parameters for this scripting system server to execute a script, this may also be needed if you wish to run a script on a specific document type such as PCB or Schematic documents.

See also

- Assigning a script to a menu, key or toolbar
- About Example Scripts.
- Check out the scripts in the **\Altium 2004\Examples\Scripts** folder to see DXP and Delphi objects and functions being used in scripts.

Assigning a script to a menu, key or toolbar

You have the ability to assign a script to a server menu, toolbar or hot key in DXP which makes it possible for you to run the script over a current PCB document for example. You will need to specify the full path to a project where the script resides in and specify which unit and procedure to execute the script.

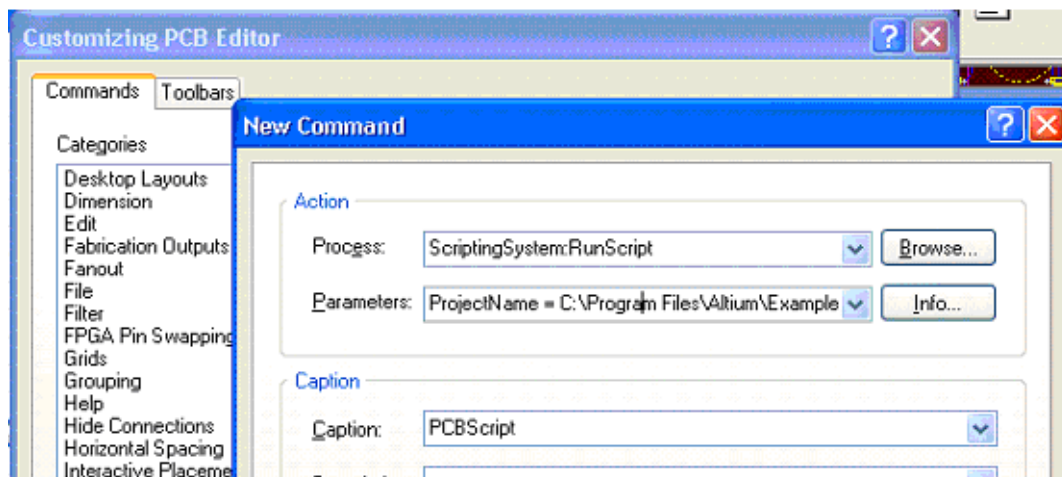
There are two parameters in this case: the **ProjectName** and the **ProcName**. For the **ProcName** parameter, you need to specify the script filename and the procedure. So the format is as follows: **ProcName = ScriptFileName>ProcedureName**. Note the GreaterThan (>) symbol used between the script file name and the procedure name.

Assigning a script to a menu example

To illustrate this ability to assign a script to a resource, we will open a PCB document in DXP and use the HelloWorld script example from the **\Scripts\VB Scripts** folder.

1. Double click on the PCB menu and the *Customizing PCB Editor* dialog appears.
2. Click on the **New** button from the *Customizing PCB Editor* dialog.

3. Choose **ScriptingSystem:RunScript** process in the **Process:** field of the *Customizing PCB Editor* dialog.
4. Enter **ProjectName = C:\Program Files\Altium2004\Examples\Scripts\VB Scripts\HelloWorld.PrjScr | ProcName = HelloWorld>ShowHelloWorldMessage** text in the **Parameters:** field for example.



5. You will need to give a name to this new command and assign a new icon if you wish. In this case, the name is **PCBScript** in the **Caption:** field of this dialog. The new commands appear in the **[Custom]** category of the **Categories** list. Click on the **[Custom]** entry from the **Categories** list. The **PCBScript** command appears in the **Commands** list of this dialog.
6. You then need to drag the new **PCBScript** command onto the PCB menu from the *Customizing PCB Editor* dialog. The command appears on the menu. You can then click on this new command and the **HelloWorld** dialog appears.

About JScript examples

The examples that follow illustrate the basic features of JScript programming in DXP. The examples show simple scripts for the DXP application.

The JScripts can use script forms, script units, functions and objects from the DXP Run Time Library and a subset of functions and objects from the Borland Delphi that is exposed in the scripting system.

The example scripts are organized into **\Examples\Scripts\JScript Scripts** folder.

Writing JScript scripts

In this section of Writing JScript scripts:

- JScript naming conventions
- Local and Global variables
- Functions
- Including comments in scripts
- Splitting a line of script.

JScript naming conventions

JScript is a case sensitive language which means the keywords, variables, function names and other identifiers must be typed with a consistent capitalization of letters.

Example

```
function Displaymessage //valid function.
Function Displaymessage // invalid function statement
a = 60 //
A = 45 // a and A are two different variables with different memory
locations!
```

Local and Global variables

Local and Global variables

Since all scripts have local and global variables, its very important to have unique variable names in your scripts within a script project. If the variables are defined outside any procedures and functions, they are global and can be accessed by any unit in the same project.

If variables are defined inside a procedure or function, then these local variables are not accessible outside these procedures/functions.

Example of local and global variables in a script

```
// Variables from UnitA script are available to this unit script,
// as long UnitA is in the same project as this Unit script.
var
    GlobalVariableFromThisUnit="Global Variable from this unit";

function TestLocal() {
var
    Local;
    // can we access a variable from UnitA without the Uses
```

JScript Reference

```
Local = "Local Variable";
ShowMessage(Local);
}

function TestGlobal {
    //ShowMessage(Local); // produces an error.
    ShowMessage(GlobalVariableFromThisUnit);
    ShowMessage(GlobalVariableFromUnitA);
}
```

Unit A script

```
Var
    GlobalVariableFromUnitA = "Global Variable from Unit A";
```

Functions

The function statement defines a JScript function. The block of the function is defined by the curly brackets {}. The syntax for a function is:

```
function functionname ([arg1 [, arg2[... , argn]]) {
    //statements
}
```

Example

```
function DisplayMessage()
{
    var
        Message = "Hello World";
    Showmessage(Message);
}
```

Returning results in a function

The return statement is used to specify the value returned by a function.

Example

```
function square (x){
    return x * x;
}
```

Parameters and Arguments

The function declaration normally has a list of parameters (with specified types but in scripts, variables are considered typeless and the scripting system works out automatically what the variable types are). The value used in place of the parameter when you make a procedure call is called an argument.

It is important to remember that functions with parameters will not appear on the Run To Item dialog

Example of a function with a parameter

```
function DisplayName (sName)
{
    ShowMessage("My Name is " + sName);
}
```

Including comments in scripts

JScript supports both C++ and C style comments. Any text between a // and the end of a line is a comment.

Any text between the characters /* and */ is a comment.

// Comment type example

```
//This whole line is a comment
```

/* */ Comment type example

```
/* This whole line is a comment */
/*
This whole line is a comment
This whole line is a comment
This whole line is a comment
*/
```

Comments can also be included on the same line as executed code. For example, everything after the semi colon in the following code line is treated as a comment.

```
ShowMessage("Hello World"); //Display Message
```

Splitting a line of script

Each code statement is terminated with the semi-colon ";" character to indicate the end of this statement. JScript allows you to write a statement on several lines of code, splitting a long instruction on two or more lines. The only restriction in splitting programming statements on different lines is that a string literal may not span several lines.

For example:

```
X.AddPoint( 25, 100);
X.AddPoint( 0, 75);
```

JScript Reference

// is equivalent to:

```
X.AddPoint( 25, 100); X.AddPoint( 0, 75);
```

But

```
"Hello World!"
```

is not equivalent to

```
"Hello
```

```
World!"
```

JScript does not put any practical limit on the length of a single line of code in a script, however, for the sake of readability and ease of debugging it is good practice to limit the length of code lines so that they can easily be read on screen or in printed form.

If a line of code is very long, you can break this line into multiple lines and this code will be treated by the JScript interpreter as if it were written on a single line.

Unformatted code example

```
If Not (PcbApi_ChooseRectangleByCorners(BoardHandle,"Choose first  
corner","Choose final corner",x1,y1,x2,y2)) Then Exit;
```

Formatted code example

```
If Not (PcbApi_ChooseRectangleByCorners(BoardHandle,  
                                         "Choose first corner",  
                                         "Choose final corner",  
                                         x1,y1,x2,y2)) Then Exit;
```

Using DXP Object models in scripts

The biggest feature of the scripting system, is that the Interfaces of DXP objects are available to use in scripts. For example you have the ability to massage design objects on Schematic and PCB documents through the use of Schematic Interfaces and PCB interfaces.

Normally in scripts, there is no need to instantiate an interface, you just extract the interface representing an existing object in DXP and from this interface you can extract embedded or aggregate interface objects and from them you can get or set property values. Interface names as a convention have an I added in front of the name for example **IPCB_Board** represents an interface for an existing PCB document in DXP.

Thus to have access to a schematic document, you invoke the **SchServer** function.

Example

```
// Checks if the current document is a Schematic document
if SchServer != Null {
    CurrentSheet = SchServer.GetCurrentSchDocument;
    if (CurrentSheet != Null {
        // statements
    }
}
```

To have access to a PCB document, you invoke the **PCBServer**.

Creating a PCB object example

```
function ViaCreation() {
var Board; //IPCB_Board;
var Via;   //IPCB_Via;
    Board = PCBServer.GetCurrentPCBBoard;
    if (Board != Null)
    {
        /* Create a Via object */
        Via          = PCBServer.PCBObjectFactory(eViaObject, eNoDimension,
eCreate_Default);
        Via.X        = MilsToCoord(7500);
        Via.Y        = MilsToCoord(7500);
        Via.Size     = MilsToCoord(50);
        Via.HoleSize = MilsToCoord(20);
        Via.LowLayer = eTopLayer;
        Via.HighLayer = eBottomLayer;
        /* Put this via in the Board object*/
    }
```

JScript Reference

```
        Board.AddPCBObject (Via) ;  
    }  
}
```

Objects, Interfaces, functions and types in your scripts can be used from the following:

- Client API
- PCB Server API
- Schematic Server API
- Work Space Manager Server API
- Nexus API
- DXP RTL functions
- Parametric processes.

JScript keywords

In this section of JScript Keywords includes the Reserved words in JScript

Reserved words in JScript

The following words are reserved in JScript and cannot be used for variable names.

B, C

abstract, boolean, break, byte, case, catch, class, const, continue

D, E

debugger, default, delete, do, else, enum, export, extends

F, G

false, final, finally, for, function, goto

I

if, implements, import, in, instanceof, int, interface, long

N, R, S

native, new, null, return, short, static, super, switch, synchronized

T, V

this, throw, throws, transient, true, try, typeof, var, void, volatile

W

while, with

JScript Statements

In this section of JScript Statements includes:

- Conditional statements
- Expressions and Operators

Conditional statements

The main conditional statements supported by the JScript;

- if (expression) else
- else if
- switch
- while loop
- do while loop
- for loop
- for in

You have to be careful to code your scripts to avoid infinite loops, ie the conditions will eventually be met.

The if.. else statement

The syntax is

```
if Condition {  
}  
else if ANotherCondition {  
}  
else{  
}
```

The switch statement

The switch statement is like a multi-way branch. The basic syntax is;

```
switch (n) {  
case 1: //n == 1  
//execute code.  
break  
case 2: //n==2  
//execute code.  
break;  
default: // if all else fails...
```

```
//execute code.
break;
}
```

The while statement

The while statement is a basic statement that allows repetitive actions until a condition is met. It is possible that the statements inside the while body never get executed if the condition is not met once.

```
while (expression) {
    //statement...
}
```

The do/while statement

The do / while statement is a variation on the while statement which is that the loop expression is tested at the bottom of the loop rather at the top. This means the body of the loop is always executed at least once. The general syntax is;

```
do
    // statements
while (expression);
```

The for statement

The for statement provides a common loop statement with a counter variable of some kind.

```
for (initialize ; test ; increment)
    statement
```

Example

```
for (count = 0; count < 10; count ++){
    showmessage(inttostr(count));
}
```

The for/in Statement

The for/in statement provides a way to loop through the properties of an object or all the elements of an array. The for/in loop does not specify the order in which the properties of an object are assigned to the variable.

```
for (variable in object)
    //statement.
```

Expressions and Operators

An expression is a valid combination of constants, variables, literal values, operators and function results. Expressions are used to determine the value to assign to a variable, to compute the parameter of a function, or to test for a condition. Expressions can include function calls.

JScript has a number of logical, arithmetic, Boolean and relational operators. Since these operators are grouped by the order of precedence which is different to the precedence orders used by Basic, C etc. For example, the AND and OR operators have precedence compared to the relational one.

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
--	Decrement
++	Increment
-	Unary negation

Logical Operators

!	Logical Not
<	Less than
>	Greater than
<=	Less than or equal to
=>	Greater than or equal to
==	equality
!=	inequality
&&	Logical and
	Logical or
?:	conditional ternary
,	comma
===	strict equality
!==	strict inequality

Bitwise Operators

~	Bitwise Not
<<	Bitwise Left Shift
>>	Bitwise Right Shift
>>>	Bitwise Unsigned right shift
&	Bitwise And
^	Bitwise Bitwise XOR
	Bitwise OR

Operator Precedence

Operator	Description
. [] ()	Field access, array indexing, function calls, and expression grouping
++ -- ~ ! delete new typeof void	Unary operators, return data type, object creation, undefined values
* / %	Multiplication, division, modulo division
+ - +	Addition, subtraction, string concatenation
<< >> >>>	Bit shifting
< <= > >= instanceof	Less than, less than or equal, greater than, greater than or equal, instanceof
== != === !==	Equality, inequality, strict equality, and strict inequality
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
&&	Logical AND
	Logical OR
?:	Conditional
= OP=	Assignment, assignment with operation
,	Multiple evaluation

JScript Functions

JScript functions are based on built in objects such as Math object and Data object etc in which you can invoke the methods to acquire the required data.

Built in Functions

Available objects that are built in JScript language in which you can use for your scripts in DXP. These objects have methods and properties that you can invoke.

Consult the Microsoft Developers Network website, <http://msdn.microsoft.com/library/en-us/script56/html/js56jsoriJScript.asp> for further details.

Example of a built in Date object

```
function DateDemo() {
    var d , s = "Today's date is: ";           //Declare variables
    d = new Date();                             //Create Date object
    s += (d.getMonth() + 1) + "/";             //Get month
    s += d.getDate() + "/";                    //Get day
    s += d.getFullYear();                       //Get year
    showMessage(s);                             //Display the date.
}
```

Built in Objects Table

Language Element	Description
ActiveXObject Object	Enables and returns a reference to an Automation object
Array Object	Provides support for creation of arrays of any data type
arguments Object	An object representing the arguments to the currently executing function, and the functions that called it
Boolean Object	Creates a new Boolean value.
Date Object	Enables basic storage and retrieval of dates and times.
Debug Object	An intrinsic object that can send output to a script debugger.
Enumerator Object	Enables enumeration of items in a collection.
Error Object	An object that contains information about errors that occur while JScript code is running.
Function Object	Creates a new function.

Language Element	Description
Global Object	An intrinsic object whose purpose is to collect global methods into one object.
Math Object	A intrinsic object that provides basic mathematics functionality and constants.
Number Object	An object representation of the number data type and placeholder for numeric constants.
Object Object	Provides functionality common to all JScript objects.
RegExp Object	Stores information on regular expression pattern searches.
Regular Expression Object	Contains a regular expression pattern.
String Object	Allows manipulation and formatting of text strings and determination and location of substrings within strings
VBArray Object	Provides access to Visual Basic safe arrays.

Forms and Components

Although Forms and Components are based on Borland Delphi's Visual Component Library, you still use the **Tool Palette** to drop controls on a form and generate JScript based event handlers and write code in JScript language.

In this section of Forms and Components:

- Components
- Designing Script Forms
- Writing Event Handlers.

Introduction to Components

The scripting system handles two types of components: Visual and Nonvisual components. The visual components are the ones you use to build the user interface, and the nonvisual components are used for different tasks such as these Timer, OpenFileDialog and MainMenu components. You use the Timer nonvisual component to activate specific code at scheduled intervals and it is never seen by the user. The Button, Edit and Memo components are visual components for example.

Both types of components appear at design time, but non visual components are not visible at runtime. Basically components from the **Tool Palette** panel are object orientated and all these components have the three following items:

- Properties
- Events
- Methods

A property is a characteristic of an object that influence either the visible behaviour or the operations of this object. For example the Visible property determines whether this objet can be seen or not on a script form.

An event is an action or occurrence detected by the script. In a script the programmer writes code for each event handler which is designed to capture a specific event such as a mouse click.

A method is a procedure that is always associated with an object and define the behavior of an object.

All script forms have one or more components. Components usually display information or allow the user to perform an action. For example a Label is used to display static text, an Edit box is used to allow user to input some data, a Button can be used to initiate actions.

Any combination of components can be placed on a form, and while your script is running a user can interact with any component on a form, it is your task, as a programmer, to decide what happens when a user clicks a button or changes a text in an Edit box.

The Scripting system supplies a number of components for you to create complex user interfaces for your scripts. You can find all the components you can place on a form from the Toolbox palette.

To place a component on a form, locate its icon on the **Tool Palette** panel and double-click it. This action places a component on the active form. Visual representation of most components is set with their set of properties. When you first place a component on a form, it is placed in a default position,

with default width and height however you can resize or re-position this component. You can also change the size and position later, by using the Object Inspector.

When you drop a component onto a form, the Scripting system automatically generates code necessary to use the component and updates the script form. You only need to set properties, put code in event handlers and use methods as necessary to get the component on the form working.

Designing Script Forms

A script form is designed to interact with the user within the DXP environment. Designing script forms is the core of visual development in the DXP. Every component you place on a script form and every property you set is stored in a file describing the form (a DFM file) and has a relationship with the associated script code (the VBS file). Thus for every script form, there is the VBS file and the corresponding DFM file.

When you are working with a script form and its components, you can operate on its properties using the *Object Inspector* panel. You can select more than one component by shift clicking on the components or by dragging a selection rectangle around the components on this script form. A script form has a title which is the **Caption** property on the *Object Inspector* panel.

Creating a new script form

With a project open in DXP, right click on a project in the *Projects* panel, and a pop up menu appears, click on the **Add New to Project** item, and choose **Script Form** item. A new script form appears with the Form1 name as the default name.

Displaying a script form

In a script, you will need to have a routine that displays the form when the script form is executed in DXP. Within this routine, you invoke the **ShowModal** method for the form. The **Visible** property of the form needs to be false if the **ShowModal** method of the script form is to work properly.

ShowModal example

```
function RunDialog() {
    DialogForm.ShowModal;
}
```

The **ShowModal** example is a very simple example of displaying the script form when the RunDialog from the script is invoked. Note, you can assign values to the components of the **DialogForm** object before the DialogForm.ShowModal is invoked.

ModalResult example

```
function bOKButtonClick(Sender) {
    ModalResult = mrOK
}

function bCancelButtonClick(Sender) {
    ModalResult = mrCancel
}
```

```
}

function RunShowModalExample() {
    //Form's Visible property must be false for ShowModal to work properly.
    if (Form.ShowModal == mrOk)      ShowMessage("mrOk");
    if (Form.ShowModal == mrCancel)  ShowMessage("mrCancel");
}
```

The **ModalResult** property example here is a bit more complex. The following methods are used for buttons in a script form. The methods cause the dialog to terminate when the user clicks either the OK or Cancel button, returning mrOk or mrCancel from the ShowModal method respectively.

You could also set the ModalResult value to mrOk for the OK button and mrCancel for the Cancel button in their event handlers to accomplish the same thing. When the user clicks either button, the dialog box closes. There is no need to call the Close method, because when you set the ModalResult method, the script engine closes the script form for you automatically.

Note, if you wish to set the form's ModalResult to cancel, when user presses the **Escape** key, simply enable the Cancel property to True for the Cancel button in the Object Inspector panel or insert `Sender.Cancel := True` in the form's button cancel click event handler.

Accepting input from the user

One of the common components that can accept input from the user is the EditBox component. This EditBox component has a field where the user can type in a string of characters. There are other components such as masked edit component which is an edit component with an input mask stored in a string. This controls or filters the input.

The example below illustrates what is happening, when user clicks on the button after typing something in the edit box. That is, if the user did not type anything in the edit component, the event handler responds with a warning message.

```
function TScriptForm.ButtonClick(Sender) {
    if (Edit1.Text == "") {
        ShowMessage('Warning - empty input!')
        return;
    }
    ' do something else for the input
}
```

Note, A user can move the input focus by using the Tab key or by clicking with the mouse on another control on the form.

Responding to events

When you press the mouse button on a form or a component, DXP sends a message and the Scripting System responds by receiving an event notification and calling the appropriate event handler method.

Writing Event Handlers

Each component, beside its properties, has a set of event names. You as the programmer decide how a script will react on user actions in DXP. For instance, when a user clicks a button on a form, DXP sends a message to the script and the script reacts to this new event. If the **OnClick** event for a button is specified it gets executed.

The code to respond to events is contained in event handlers. All components have a set of events that they can react on. For example, all clickable components have an **OnClick** event that gets fired if a user clicks a component with a mouse. All such components have an event for getting and loosing the focus, too. However if you do not specify the code for **OnEnter** and **OnExit** (**OnEnter** - the control has focus; **OnExit** - the control loses focus) the event will be ignored by your script.

Your script may need to respond to events that might occur to a component at run time. An event is a link between an occurrence in DXP such as clicking a button, and a piece of code that responds to that occurrence. The responding code is an event handler. This code modifies property values and calls methods.

List of properties for a component

To see a list of properties for a component, select a component and in the **Object Inspector**, activate the **Properties** tab.

List of events for a component

To see a list of events a component can react on, select a component, and in the **Object Inspector** activate the **Events** tab. To create an event handling procedure, decide on what event you want your component to react, and double click the event name.

For example, select the **Button1** component from the **Toolbox** panel and drop it on the script form, and double click next to the **OnClick** event name. The scripting system will bring the Code Editor to the top of the DXP and the skeleton code for the **OnClick** event will be created.

For example, a button has a **Close** method in the **CloseClick** event handler. When the button is clicked, the button event handler captures the on click event, and the code inside the event handler gets executed. That is, the **Close** method closes the script form.

Event Handler Example

```
function bCloseClick(Sender)
{
    Close();
}
```

Standalone function example

```
function DrawSine()
{
    ShowModal();
}
```

JScript Reference

In a nutshell, you just select a component, either on the form or by using the **Object Inspector** panel, select the **Events** page, and double click on the right side of the **OnClick** event, a new event handler will appear on the script. OR double click on the button and the scripting system will add a handler for this **OnClick** event. Other types of components will have completely different default actions.

List of methods for a component

To see a list of methods for a component, see the Components Reference and check out the Borland Delphi documentation for many of these components.

Using Components in Script Forms

Dropping components on a script form

To use components from the Tool Palette panel in your scripts, you need to have a script form first before you can drop components on the form. Normally when you drop components on a script form, you do not need to create or destroy these objects, the script form does them for you automatically.

The scripting system automatically generates code necessary to use the component and updates the script form. You then only need to set properties, put code in event handlers and use methods as necessary to get the script form working in DXP.

Creating components from a script

You can also directly create and destroy components in a script – normally you don't need to pass in the handle of the form because the script form takes care of it automatically for you, thus you just normally pass a Nil parameter to the Constructor of a component.

For example, you can create and destroy Open and Save Dialogs (TOpenDialog and TSaveDialog classes as part of Borland Delphi Run Time Library).

Index

A

About JScript examples	6
Adding JScripts to a project	3
Assigning a script to a menu_ key or toolbar	5

B

Built in JS Functions	18
-----------------------------	----

C

Components for JScripts	20
Conditional statements	14
Creating new JScripts	3

D

Designing Script Forms	21
------------------------------	----

E

Executing a JScript in DXP	3
Exploring the JScript language	1
Expressions and Operators	16

F

Forms and Components	20
Functions	8

I

Including comments in scripts	9
Introduction	1

J

Javascript naming conventions	7
JScript functions	18
JScript keywords	13
JScript source files	2
JScript Statements	14

L

Local and Global variables	7
----------------------------------	---

R

Reserved words in Javascript	13
------------------------------------	----

S

Splitting a line of script	9
----------------------------------	---

U

Using DXP Object models in scripts	11
--	----

W

Writing Event Handlers	23
Writing JScript scripts	7

Revision History

Date	Version No.	Revision
01-Dec-2004	1.0	New product release

Software, hardware, documentation and related materials:

Copyright © 2004 Altium Limited.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of Altium Limited. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium, CAMtastic, Design Explorer, DXP, LiveDesign, NanoBoard, Nexar, nVisage, P-CAD, Protel, Situs, TASKING and Topological Autorouting and their respective logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.