Table of Contents

1 Introduction

The MacroScript SDK is the easiest way to add a macro language to your applications.

Seamlessly add the full power and simplicity of the Macro Scheduler script language to your own applications. Run Macro Scheduler code directly within your application without having to shell to a Macro Scheduler script file or executable. Your users will not need to have Macro Scheduler installed.

For support contact support@mjtnet.com or visit http://www.mjtnet.com/contact.htm

2 ActiveX Reference

Registering mscript.ocx

Before using the OCX you will need to register it. Start a command prompt, cd to the directory containing mscript.ocx and then RegSvr32. E.g.:

```
cd c:\macroscript_sdk
regsvr32 mscript.ocx
```

The OCX will need to be registered on all PCs that will use the SDK. Your application's setup program can do this on installation.

The mscript.dll file should be available on the path. It could be in the same folder as your application or in the system path. For development you may prefer to put it in the system path (e.g. \Windows\System32) so that your ID can find it.

Library: Mscript
Interface: MacroScript

Properties

script
parms

Methods

Init
Cleanup
RunCode
RunScript

Run
GetVar
Stop

Deployment

Deployment

2.1 MScript.MacroScript

Properties

script
parms

Methods

Init
Cleanup
RunCode
RunScript
Run
GetVar
Stop

2.1.1 script

Property Get Script() As String
Property Let Script(RHS As String)

Set script to the code to be executed by the Run method.

2.1.2 parms

Property Get parms() As String
Property Let parms(RHS As String)

Set parms to the variables to be passed into the script via the Run method.

2.1.3 Init

Sub Init

Initializes the script interpreter and variable table. Must always be called before calling RunCode, RunScript, Run or GetVar.

2.1.4 Cleanup

Sub Cleanup

Must always be called after calling RunCode, RunScript, Run.

2.1.5 RunCode

Function RunCode(ByVarl thecode As String, ByVal parms As String) As String

Executes script code contained in *thecode* and returns the value of the MACRO_RESULT script variable.

To pass variables into the script specify variables in *parms* as if they are passed on the command line. E.g.:

/forename=fred /surname=bloggs /age=46

Init should be called before calling RunCode. Call Cleanup afterwards.

2.1.6 RunScript

Function RunScript(ByVarl thefile As String, ByVal parms As String) As String

Executes script file *thefile* and returns the value of the MACRO_RESULT script variable.

To pass variables into the script specify variables in *parms* as if they are passed on the command line. E.g.:

/forename=fred /surname=bloggs /age=46

Init should be called before calling RunCode. Call Cleanup afterwards.

2.1.7 Run

Function Run() As String

Executes the code set in the script property.

To pass variables into the script specify variables in the parms property as if they are passed on the command line. E.g.:

/forename=fred /surname=bloggs /age=46

Init should be called before calling RunCode. Call Cleanup afterwards.

2.1.8 GetVar

Function GetVar(ByVal varname As String) As String

Retrieves the value of a script variable. This can be called after Init, Run, RunCode or RunScript. If called after Cleanup it will return nothing since Cleanup empties the variable table and frees the script object.

2.1.9 Stop

Sub Stop

Stops any running script or script code.

2.2 Deployment

mscript.dll and mscript.ocx must be deployed with your application. Remember to register mscript.ocx as above.

Optionally, for text capture support, include the following files:

GetWordNT.dll
ICall.dll

All files need to be in the path. They could be stored in the same folder as your application or in the system path (e.g. \Windows\System32).

3 DLL Reference

Functions

Init
Cleanup
RunCode
RunScript
GetVar

Deployment

3.1 Init

Syntax:

function Init: integer; stdcall;

__stdcall int Init(VOID);

Initialises the script object and variable table.

Returns:

Returns zero if successful.

Init should be called before RunCode or RunScript.

3.2 Cleanup

Syntax:

function Cleanup: integer; stdcall;

__stdcall int Cleanup(VOID);

Should be called when all operations are complete (e.g. after RunCode, RunScript or GetVar) to empty the variable table and free the script object.

Returns:

Returns zero.

3.3 RunCode

Syntax:

function RunCode(thecode, parms, res: pchar, rescount: integer): integer; stdcall;

```
__stdcall int RunCode(
  LPTSTR thefile,
  LPTSTR parms,
  LPTSTR res,
  int rescount;
);
```

Parameters:

thecode: string containing the MacroScript code to execute
parms: a command line to pass into the script
res: buffer for value of MACRO_RESULT variable.
rescount: length of buffer.

Returns:

The function returns the number of characters written to *res*.

Remarks

Init should be called before calling RunCode.

3.4 RunScript

Syntax:

function RunScript(thefile, parms, res: pchar; rescount: integer): integer; stdcall;

__stdcall int RunScript(
* LPTSTR thefile,*
* LPTSTR parms,*
* LPTSTR res,*
* int rescount*
);

Parameters*:*

thefile: filename of a script file containing MacroScript code
parms: a command line to pass into the script
res: buffer for value of MACRO_RESULT variable.
rescount: length of buffer.

Returns*:*

The function returns the number of characters written to *res*.

Remarks

Init should be called before calling RunScript.

3.5 GetVar

Syntax:

function GetVar(varname, value: pchar; valcount: integer): integer; stdcall;

__stdcall int GetVar(
* LPTSTR varname,*
* LPTSTR value,*
* int valcount;*
);

Parameters:

varname: the name of a variable whose value should be retrieved.
value: the buffer into which the value of variable is written
valcount: length of buffer

Returns:

The number of characters written to *value*.

Remarks:

GetVar can be used to retrieve the value of a variable at any point after calling Init and before calling Cleanup.

3.6 Deployment

mscript.dll must be deployed with your application.

Optionally, for text capture support, include the following files:

GetWordNT.dll
ICall.dll