DATATON SMARTSCRIPT
User's Guide

Version 1.0

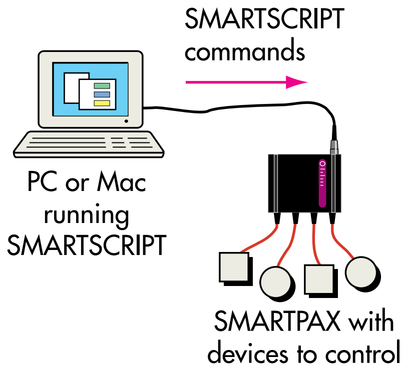
dataton

TRUE MULTIMEDIA

Table of Contents

Document number: 3949

1 INTRODUCTION


SMARTSCRIPT commands
PC or Mac running SMARTSCRIPT
SMARTPAX with devices to control

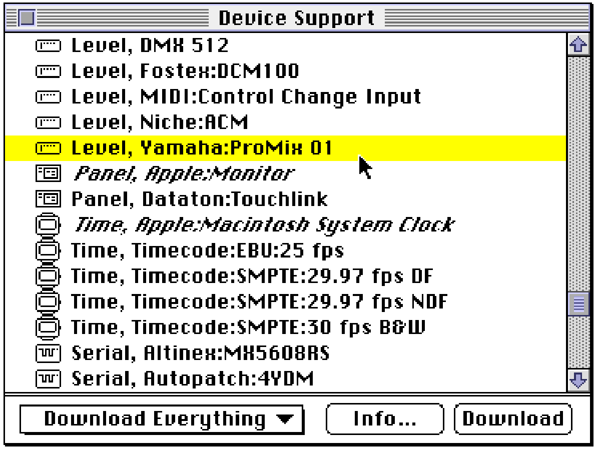
Dataton SMARTSCRIPT allows you to control external devices from any supported software application and from custom written software. It is based on industry standards such as ActiveX for Microsoft Windows 95/NT and Xtra for Macromedia Director and Authorware. Your computer talks to the external devices through Dataton SMARTPAX – an intelligent, modem-sized box specifically designed for multimedia device control. Software drivers downloaded into SMARTPAX handle the communication protocol required for each device, thereby offloading the host computer.

Software drivers are available for over 200 models of devices, such as audio video, tape and discs players, matrix switchers, slide and video projectors, production switchers, computer graphics stations, lighting, motors, animatronics, etc. Based on many years of experience, those software drivers contain all the detailed knowledge needed for reliable device control.

SMARTPAX connects to the serial port on your computer. Each SMARTPAX controls up to four devices or sub-systems. The system is easily expanded by daisy-chaining additional SMARTPAX units. The physical connection between SMARTPAX and the device to be controlled is a smartlink cable, which provides the correct signal and mating connectors. Smartlink cables are available for a variety of interfaces, ranging from simple switches and analog outputs all the way up to infrared, timecode and various kinds of serial data (RS232, RS422, DMX-512 and MIDI).

SMARTSCRIPT Capabilities

Using SMARTSCRIPT, you can control all devices listed in the Device Support window in TRAX. The Device Support window in TRAX is the best, most up to date, list of supported devices.



Device Support

Level, DMX 512
Level, Fostex:DCM100
Level, MIDI:Control Change Input
Level, Niche:ACM
Level, Yamaha:ProMix 01
Panel, Apple:Monitor
Panel, Dataton:Touchlink
Time, Apple:Macintosh System Clock
Time, Timecode:EBU:25 fps
Time, Timecode:SMPTE:29.97 fps DF
Time, Timecode:SMPTE:29.97 fps NDF
Time, Timecode:SMPTE:30 fps B&W
Serial, Altinex:MX5608RS
Serial, Autopatch:4YDM

TRAX has built-in functions for creating new drivers for devices that can be controlled by serial data (RS232, RS422, MIDI, etc). Such device drivers can then be used from SMARTSCRIPT, just like the standard drivers.

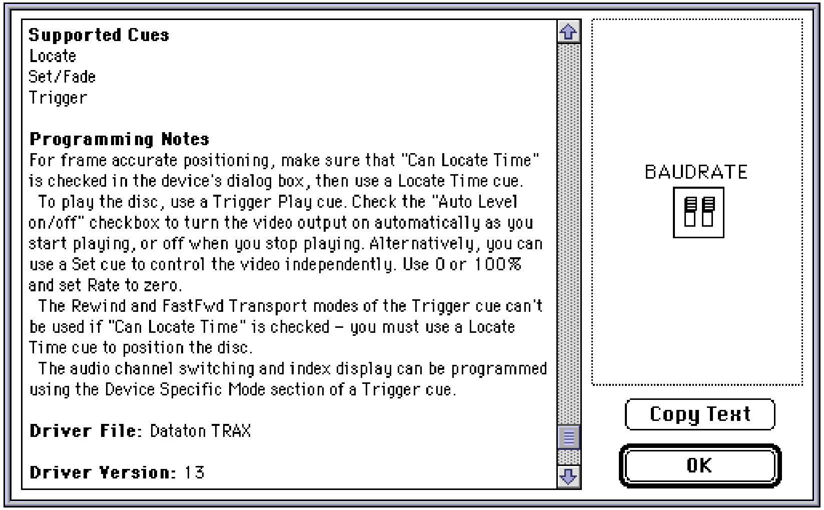
A separate tool is available for creating drivers for devices that can be controlled using an infrared remote control. Contact your Dataton dealer for further information about devices and drivers.

SMARTSCRIPT Cues

The various CueXxx commands in SMARTSCRIPT mimic the corresponding cues in TRAX. You will need a basic knowledge of TRAX in order to use SMARTSCRIPT. If you're not familiar with TRAX, you should at least read chapter seven in the TRAX 3 handbook, which introduces the various cue types.

TRAX has a built-in database with details for each supported device. This tells you how to configure the device and which smartlink cable you need to connect it to SMARTPAX.

◆ **NOTE:** The TRAX 3 application program and handbook are available free of charge under the “Free Software” heading at www.dataton.com.



Supported Cues
Locate
Set/Fade
Trigger

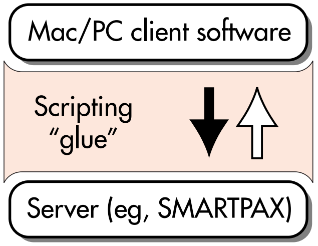
Programming Notes
For frame accurate positioning, make sure that "Can Locate Time" is checked in the device's dialog box, then use a Locate Time cue.
 To play the disc, use a Trigger Play cue. Check the "Auto Level on/off" checkbox to turn the video output on automatically as you start playing, or off when you stop playing. Alternatively, you can use a Set cue to control the video independently. Use 0 or 100% and set Rate to zero.
 The Rewind and FastFwd Transport modes of the Trigger cue can't be used if "Can Locate Time" is checked – you must use a Locate Time cue to position the disc.
 The audio channel switching and index display can be programmed using the Device Specific Mode section of a Trigger cue.

Driver File: Dataton TRAX

Driver Version: 13

The database also tells you which cues you use to control the device. The SMARTSCRIPT cue commands follow the same naming conventions as TRAX. For example, the Locate Time cue in TRAX corresponds to the CueLocateTime command in SMARTSCRIPT.

Scripting Overview



Scripting allows you to control SMARTPAX or Dataton TRAX from your own software. Building software solutions from already existing components saves time and makes it possible to enhance one piece of software without affecting the other. The scripting language and scripting standard act as the "glue" between the pieces, allowing them to communicate.

The piece of software that provides a desired function or service is often called a "server", and the piece of software requiring the service is called a "client". The server and the client software may exist within the same computer. Alternatively, the server may run on its own computer or other dedicated hardware connected via a serial port or a network.

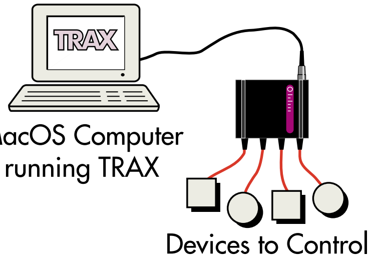
When using SMARTSCRIPT, SMARTPAX acts as a server to your application, providing control of the devices. SMARTSCRIPT sits between your application and the system bus that talks to the SMARTPAX units. This allows your application to communicate with the devices being controlled on a high level.

Likewise, TRAXSCRIPT allows your application to use TRAX as an even higher level server, providing similar capabilities as SMARTSCRIPT plus the features contributed by TRAX, such as timelines and multitasking.

Other Scripting Languages

Although SMARTSCRIPT and TRAXSCRIPT provide their own vocabularies, they are not full-fledged languages on their own. Instead SMARTSCRIPT and TRAXSCRIPT team up with the scripting language of the host platform (eg, Lingo for Macromedia products and Visual Basic for Microsoft products), essentially enhancing these languages with specialized commands and functions for controlling external devices.

Scripting versus TRAX



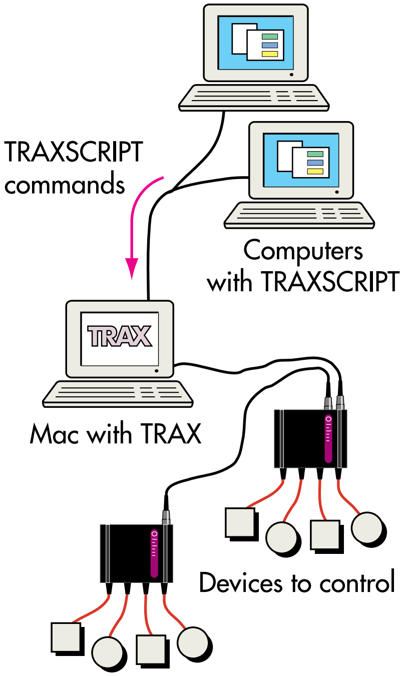
The most common way of programming a Dataton system is by using Dataton TRAX. This MacOS application program allows you to design and program your system using an intuitive, icon-based, point-and-click interface. TRAX provides the ultimate power and flexibility, with features such as:

- True multitasking, supporting a virtually unlimited number of concurrent timelines and other tasks.
- Tight synchronization, both among timelines and relative external devices.
- External inputs using push-buttons, sensors or touch-panels for fully interactive control.
- Device status continuously displayed.

However, some applications don't require those capabilities, or may need to be integrated with other pieces of software or computer standards. SMART-SCRIPT essentially replaces TRAX for running the system, and allows you to control the SMARTPAX units and associated devices using the application of your choice instead of TRAX.

◆ **NOTE:** When using SMARTSCRIPT, TRAX is still needed for the initial system configuration (see "System Configuration" on page 13). Once the system has been configured, and all device drivers downloaded to the SMARTPAX units, TRAX is no longer required to run the system.

SMARTSCRIPT versus TRAXSCRIPT



SMARTSCRIPT allows you to control all device functions directly through SMARTPAX, but it doesn't give you the capabilities provided by TRAX itself, such as multiple timelines or tight synchronization. If you need some of those capabilities, but still must be able to access them from another piece of software or another computer, you can use TRAXSCRIPT instead. This allows your application to talk to TRAX instead of directly to the SMARTPAX – thus providing all the capabilities of TRAX as well as direct device control similar to SMARTSCRIPT.

TRAXSCRIPT is particularly suitable for larger applications, where you may have multiple clients accessing a shared pool of devices. These devices are then managed through TRAX, which can be connected to the client computers using a standard TCP/IP compatible network, a serial port or through AppleScript. Just like SMARTSCRIPT, TRAXSCRIPT is also supported by an Xtra for Macromedia applications and an ActiveX for Windows applications.

To summarize, SMARTSCRIPT allows you to control the devices directly from a single computer. It provides device control based on the drivers downloaded into the SMARTPAX units, but doesn't provide the higher level capabilities that are part of TRAX. Nor does it need the TRAX computer to run the system.

TRAXSCRIPT, on the other hand, allows you to control the system from multiple computers or other devices (using a computer network, a serial data link or AppleScipt). It allows for additional interactive capabilities, such as contact closures, MIDI inputs, TOUCHLINK touch panels, wireless remote control, etc. All these control sources can be active at the same time, if desired.

The additional capabilities provided by TRAXSCRIPT over SMARTSCRIPT comes at the cost of increased system complexity and the necessity of having the TRAX computer present in the finished system.

Component Standards

In order to make software components talk to each other, a software component standard is required. This standard allows software pieces to locate each other and communicate using a common language. SMARTSCRIPT supports two such standards:

- Microsoft ActiveX, the standard for Windows software components.
- Macromedia Xtra, used in Director and Authorware.

These provide the same capabilities and are used in very similar ways. Due to the somewhat richer ActiveX standard, the ActiveX implementation provides a few alternatives for doing basic things, such as specifying which serial port to use to talk to the SMARTPAX units. However, these are provided only for the sake of convenience and adherence to the ActiveX standard. Specifically, they don't provide any additional functionality over the Xtra implementation.

Windows

The standard for Windows software components is called ActiveX. ActiveX is supported by a large number of Windows applications and programming languages such as Microsoft PowerPoint, Excel, Visual C++ and Visual Basic, as well as Borland Delphi, Asymetrix Toolbook and many others.

Most of the Microsoft applications use Visual Basic as their scripting language. Borland Delphi uses Object Pascal and Asymetrix Toolbook has its own proprietary scripting language. Although the SMARTSCRIPT statements are very simple and straightforward, you need a basic understanding of the host application's scripting language in order to use SMARTSCRIPT.

Macromedia

The standard for Macromedia software components is called Xtra. This is supported by Director, Authorware and other Macromedia applications under both MacOS and Windows. It ties into Macromedia's scripting language, called Lingo, and essentially extends Lingo with new capabilities.

After installing the SMARTSCRIPT Xtra and configuring the system, you can control virtually any external presentation device using cues in Director's Score window, or through push-buttons and other interactive controls. Although the SMARTSCRIPT statements are very simple and straightforward, you need a basic understanding of Macromedia's Lingo language in order to use SMARTSCRIPT.

◆ **NOTE:** When using Director or Authorware under Windows, you can choose either the Xtra or the ActiveX implementation of SMARTSCRIPT. However, in order to use the ActiveX implementation you need an additional Xtra-to-ActiveX adapter, available from Macromedia.

Installation

The SMARTSCRIPT software is available free of charge on the Internet from www.dataton.com under the "Free Software" heading. There you'll also find the latest revision of this handbook.

There are two versions of SMARTSCRIPT; one for MacOS and one for Windows. The MacOS version includes the SMARTSCRIPT Xtra for Macromedia running on MacOS (PowerPC only). The Windows version includes both the ActiveX and Xtra implementations for Windows.

In addition to the correct version of SMARTSCRIPT, you'll also need the following items:

- The host application of your choice (eg, Macromedia Director or Microsoft Visual Basic, Excel or PowerPoint).
- The system description file matching the desired system configuration (see "System Description File" on page 14).

- The Dataton cable to connect the computer to the SMARTPAX. For Macintosh this is the TRAX CABLE (product number 3425) and for the PC this is the PC CABLE (product number 3429).
- The required number of SMARTPAX control units, with power supplies (12V DC ADAPTOR 3334 if using SMARTPAX QC, AC PAX ADAPTOR 3337 if using SMARTPAX). Use Dataton SYSTEM CABLE to daisy-chain multiple SMARTPAX units.
- The appropriate SMARTLINK cables for connecting the devices to be controlled.

For more details on SMARTPAX, power supply and smartlink cables, please refer to the TRAX 3 handbook. SMARTPAX QC, its power supply and some additional SMARTLINK cables are described in the TRAX 3.5 addendum. Both these manuals are available in Adobe Acrobat format on the Internet at www.dataton.com under the "Free Software" heading. Alternatively, they can be ordered through your Dataton dealer.

MacOS

After obtaining "SMARTSCRIPT for MacOS", you need to install it on your computer before you can use it. Unpack the "SMARTSCRIPT for MacOS.sit" file using StuffIt Expander, or similar. This results in a folder named SMARTSCRIPT. Open this folder and move the file named "SMARTSCRIPT.Mac" to the Xtras folder, located in the same folder as your Director application. The folder also contains an example Director movie and its accompanying system description file (see "System Description File" on page 14). Make sure that the SDF file stays in the same folder as the example movie.

◆ **NOTE:** SMARTSCRIPT for MacOS runs only on PowerPC models. It does not run on 68k Macs.

Windows

After obtaining "SMARTSCRIPT for Windows", you need to install it on your computer before you can use it. Double-click the "SSWin.exe" file to unpack it. It creates a folder named SMARTSCRIPT, containing two folders named "Windows ActiveX" and "Macromedia Xtra". Open the folder containing the desired SMARTSCRIPT implementation.

For the Macromedia Xtra implementation of SMARTSCRIPT, simply move the file named "SMARTSCRIPT.x32" to the Xtras folder, located in the same folder as your Director.EXE application. The folder also contains a sample Director presentation and its accompanying system description file (see "System Description File" on page 14). Make sure that the SDF file is in the same folder as the example movie.

For the Windows ActiveX implementation of SMARTSCRIPT, double-click the Setup.EXE file. This installs the SMARTSCRIPT components as well as a sample application in a folder of your choice. This sample is written in Visual Basic, and it includes the source code as well as the finished application ready to run together with its accompanying system description file.

◆ **NOTE:** The Xtra and ActiveX implementations of SMARTSCRIPT for Windows run under Windows95, or later, as well as Windows NT 4.0, or later. They do not run under Windows version 3 (ie, only 32-bit versions are provided).

2 GETTING STARTED

This chapter shows you how to use SMARTSCRIPT with Director or Visual Basic as the host application. It assumes that SMARTSCRIPT has been installed on your computer as described under "Installation" on page 10.

System Configuration

While SMARTSCRIPT contains all functions needed to control the external devices once the system is set up, it does not include the functions required to configure the system and download the device drivers to the SMARTPAX units. SMARTSCRIPT assumes that the system is already configured properly, and uses a system description file to learn about the system configuration.

If you know the system configuration in advance, you can ask your Dataton dealer to configure the SMARTPAX units for you when you order them, providing the matching system description file and smartlink cables.

Alternatively, you can do the system configuration yourself using DATATON TRAX (MacOS only). TRAX is available free of charge on the internet at www.dataton.com. You'll also need the TRAX CABLE (product number 3425) to connect the Mac to the SMARTPAX. Please refer to chapter 5 in the TRAX 3 handbook for details on how to configure the devices. The "Device Support" section in chapter 3 of the TRAX 3 manual describes how to download the device drivers to the SMARTPAX units.

◆ **IMPORTANT:** You must add all device icons to the Device window in TRAX, and configure them as desired, before downloading the device drivers to SMARTPAX. You can not use the "Device Drivers, Manual Mode" setting in the Device Support window as this doesn't provide the required information in the system description file.

System Description File

SMARTSCRIPT uses a system description file to learn about the names and other attributes of the devices to be controlled. You must inform SMARTSCRIPT about the name of this system description file. This is done either when using the Open command, or using the SysDescFile property (ActiveX only). The name of the system description file usually ends with ".SDF".

If your Dataton dealer delivered the SMARTPAX units pre-loaded with the device drivers of your choice, a matching system description file should be included. If not, please contact your Dataton dealer for assistance.

First Floor Kiosk

First Floor Kiosk.SDF

If you configured your SMARTPAX units yourself, as described above, you must save the TRAX show file and create the system description file before quitting TRAX. To do so, choose "Save" on the File menu while holding down the Shift key. (This applies to TRAX version 3.5.2 or later. TRAX version 3.5.1 uses the Option key instead of the Shift key.) Doing so saves the TRAX show file and generates the system description file at the same time.

The system description file has the same name as the show file, but ends with ".SDF". For example, if you save the show using the name "First Floor Kiosk" TRAX will also create a file named "First Floor Kiosk.SDF" in the same folder as the show file.

- ◆ **NOTE:** If you forget to hold down the Shift key when choosing the Save command, you must make a small change to the show before you can choose the Save command again. Just moving a device icon a few pixels is sufficient to re-enable the Save command.

- ◆ **IMPORTANT:** Keep the TRAX show file if you later need to re-configure the system, for example in order to add more devices. TRAX can not read the system description (SDF) file.

The resulting system description file contains all information needed by SMARTSCRIPT to control the devices. If you make any changes in TRAX that require a new download to the SMARTPAX units, you must also re-generate the system description file by holding down the Shift key when saving.

The system description file is a plain text file that can be opened using Simple-Text or any other word processor. Do not make any changes in this file. If you do, SMARTSCRIPT may not be able to read it, or some devices may not work properly.

In order to use the system description file, it must reside on a disk accessible to SMARTSCRIPT. For example, if you're using SMARTSCRIPT on a PC, you must first transfer the system description file to the PC. If your computers are on a network, you may be able to transfer the file using the network. If not, put the file on a PC formatted floppy disk and transfer it manually to the PC.

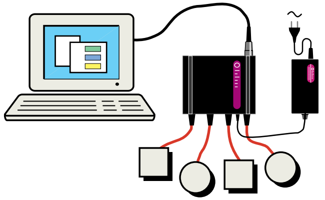
◆ **NOTE:** In order to recognize PC-formatted disks, you must have the "PC Exchange" control panel, or equivalent, installed on your Macintosh.

Macromedia

This is a step-by-step description of how to create a simple SMARTSCRIPT application using Macromedia Director.

Before starting, make sure you have installed SMARTSCRIPT on your computer as described on "Installation" on page 10, and that you have the system description file matching your configuration. Alternatively, if you just want to check out the software, use the example system description file named "SSTest.SDF", included with SMARTSCRIPT. This file must be located in the same folder as the Director movie.

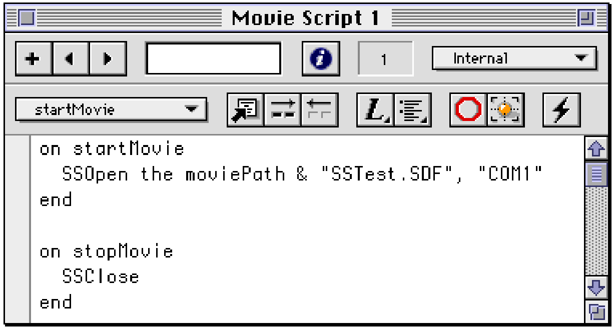
Hardware Hook-up



Connect the first SMARTPAX to the a free serial port. If you're using a PC then you need the Dataton PC CABLE (product number 3429) connected to a free COM-port. On the Mac, use a Datatoon TRAX CABLE (product number 3425) connected to the Modem or Printer port. Connect the appropriate power supply to the SMARTPAX, and connect the devices using the appropriate smartlink cables.

Activating SMARTSCRIPT from Director

Start Director and open the Script window. Enter the startMovie and stopMovie handlers, as shown below.



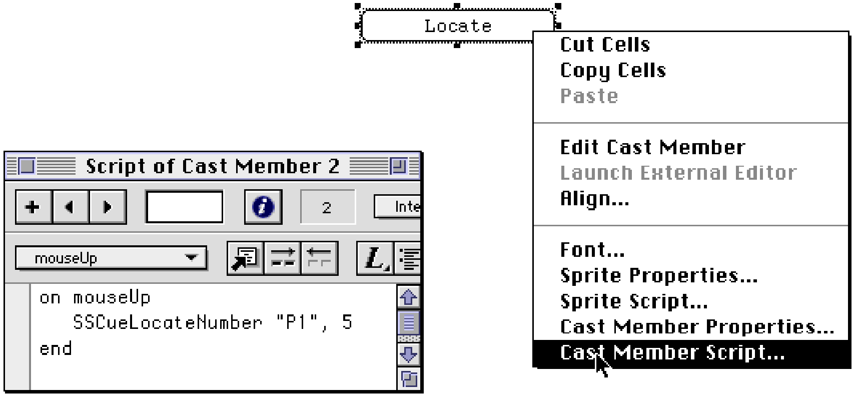
```
on startMovie
  SSOpen the moviePath & "SSTest.SDF", "COM1"
end

on stopMovie
  SSClose
end
```

Choose Save on the File menu. Save the movie in the same folder as the system description file you want to use.

◆ **IMPORTANT:** You must save the movie to disk before you attempt to run it. The startMovie script above uses "the moviePath" to locate the SDF file. This feature doesn't work properly until the movie has been saved to disk.

Adding a Button to the Movie

Choose “Control: Push Button” on the Insert menu. Type “Locate” into the button. Right-click the button (Control-click on the Mac) and choose “Cast Member Script” on the contextual menu.



```
on mouseUp
  SSCueLocateNumber "P1", 5
end
```

Enter the script as shown above into the case member script for the button.

Open the Message window by choosing “Message” on the Window menu. Click the Play button in Director’s Control Panel to run the movie. SMARTSCRIPT should now send a command to tell device “P1” to go to position five.

◆ **NOTE:** For the above to work, it is assumed that the system configuration indeed includes a device named “P1” that is capable of locating to a numeric position. If that’s not the case, SMARTSCRIPT will display an error message when you attempt to run the script.

In order to really see something happening, you must of course have the actual device connected, as described under “Hardware Hook-up” on page 16. If any error occurs, this will be indicated in the Message window.

◆ **IMPORTANT:** Keep the Message window open while working with SMARTSCRIPT. If the Message window is closed, errors may go unnoticed. See “Error Codes” on page 70 for more details on how to handle errors.

Where to Go from Here

Read the reference chapter, page 32, to learn more about the various cue types and other scripting commands. Take a look at the enclosed example Director movie, which demonstrates most commands. If you’re new to Director and Lingo, it could be a good idea to read one of the many book available on how to program in Lingo.

Windows

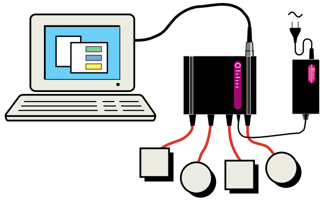
This is a step-by-step description of how to create a simple SMARTSCRIPT application using Microsoft Visual Basic (version 5.0 or later). Many other applications – particularly those from Microsoft – have a programming environment almost identical to that of Visual Basic.

If you use another programming language, the SMARTSCRIPT cues, commands and properties should be virtually identical to those presented by Visual Basic. Other language constructs, and the procedure required to create programs, will be different, though. Please refer to the appropriate documentation for your programming language for details on how to use ActiveX controls.

Before starting, make sure you have installed SMARTSCRIPT for Windows as described under "Installation" on page 10, and that you have the system description file matching your configuration. Alternatively, if you just want to check out the software, use the example system description file named "SSTest.SDF", included with SMARTSCRIPT.

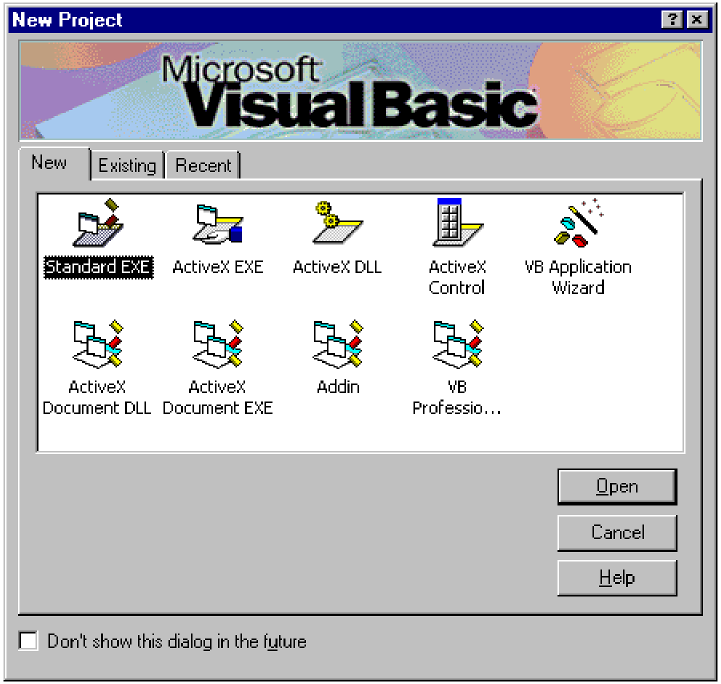
Hardware Hook-up

Connect the first SMARTPAX to a serial port on the PC using the Dataton PC CABLE. Connect the appropriate power supply to the SMARTPAX, and connect the devices using the appropriate smartlink cables.



Creating the Visual Basic Project

Start Visual Basic and create a new project of type "Standard EXE".

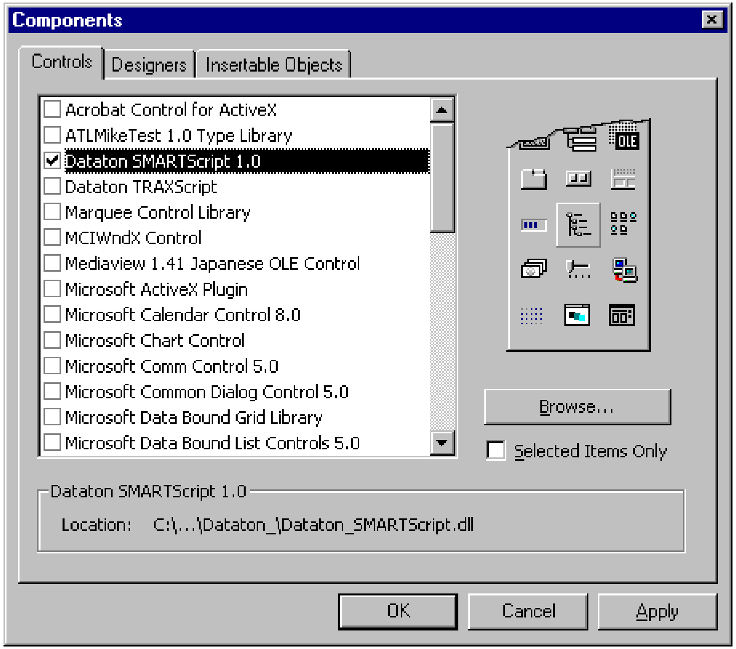


Adding SMARTSCRIPT to the Toolbox

This step will add SMARTSCRIPT to the Visual Basic toolbox, making it available for use in your project.

- Make sure that the toolbox is visible. If not, choose "Toolbox" on the View menu to display it.

- Choose “Components…” on the Project menu to display the list of available ActiveX controls.
- Make sure that “Dataton SMARTSCRIPT” is checked in the list.



- [ ] Acrobat Control for ActiveX
- [ ] ATLMikeTest 1.0 Type Library
- [✓] Datation SMARTScript 1.0
- [ ] Dataton TRAXScript
- [ ] Marquee Control Library
- [ ] MCIWndX Control
- [ ] Mediaview 1.41 Japanese OLE Control
- [ ] Microsoft ActiveX Plugin
- [ ] Microsoft Calendar Control 8.0
- [ ] Microsoft Chart Control
- [ ] Microsoft Comm Control 5.0
- [ ] Microsoft Common Dialog Control 5.0
- [ ] Microsoft Data Bound Grid Library
- [ ] Microsoft Data Bound List Controls 5.0

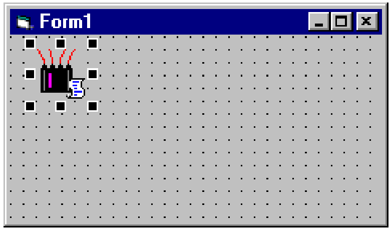
[ ] Selected Items Only

Dataton SMARTScript 1.0
Location: C:\…\Dataton_\Dataton_SMARTScript.dll

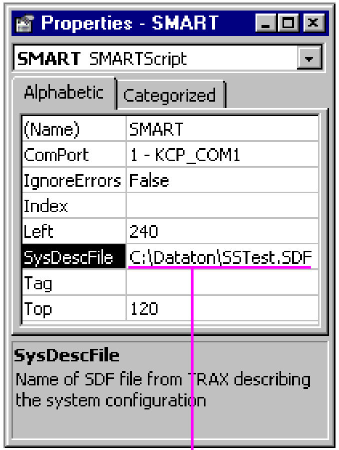
◆ **NOTE:** If “Datation SMARTSCRIPT” doesn’t appear in the list, you have not installed SMARTSCRIPT properly. See “Installation” on page 10.

Adding SMARTSCRIPT to the Main Form

In order to use SMARTSCRIPT, its icon must appear on an active form. Visual Basic automatically creates a form named "Form1" when you create a new "Standard EXE" project. To add the SMARTSCRIPT icon to the form, first select the SMARTSCRIPT icon in the Toolbox, then draw the icon on the form.



Configuring SMARTSCRIPT



SMART SMARTScript

(Name)	SMART
ComPort	1 - KCP_COM1
IgnoreErrors	False
Index	
Left	240
SysDescFile	C:\Dataton\SSTest.SDF
Tag	
Top	120

SysDescFile
Name of SDF file from TRAX describing the system configuration

Enter the full path and file name for your system description file here.

Make sure that the SMARTSCRIPT icon is still selected in the Form window, as indicated by the black selection handles around its icon (see above). Then choose "Properties Window" on the View menu to display the list of SMART-SCRIPT properties.

If you don't see the SMARTSCRIPT properties in the Properties window, then click the SMARTSCRIPT icon on the form so its selection handles appear.

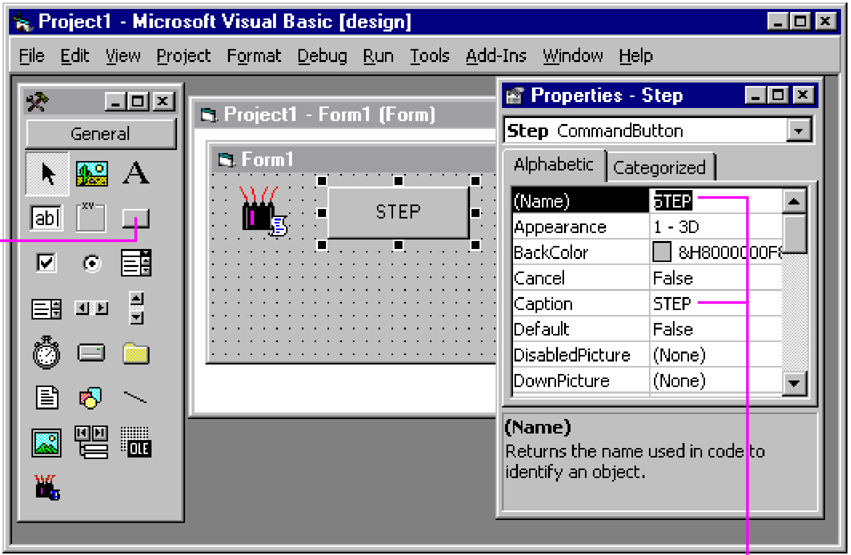
Set the Name property to "SMART", as shown to the left. Specify the serial port to which you connected the SMARTPAX as the ComPort property by first clicking the property and then choosing from the drop-down menu. Typically, you would connect SMARTPAX to the serial port named COM1, in which case you would choose "1 - KCP_COM1" on the dropdown menu.

Enter the full path and file name of the system description file in the SysDescFile property. SMARTSCRIPT will tell you if it can't find the system description file. The example system description file "SSTest.SDF" is located in the folder you specified when you installed SMARTSCRIPT (see "Setup.EXE" under "Windows" on page 12).

Adding a Command Button to the Form

Choose the CommandButton icon in the Toolbox and draw a button on a form. Set the Name and Caption properties of the button to STEP in the Properties window.

Command button.


Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Tools Add-Ins Window Help
General
Project1 - Form1 (Form)
Form1
STEP
Properties - Step
Step CommandButton
Alphabetic Categorized

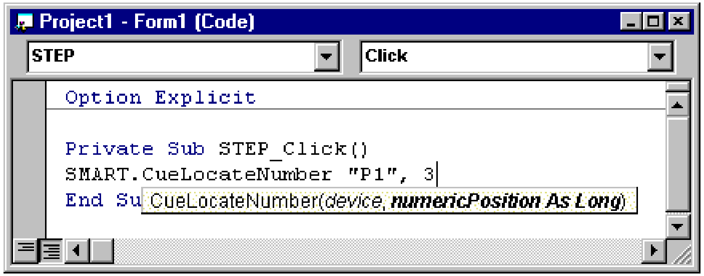
(Name)	STEP
Appearance	1 - 3D
BackColor	&H8000000F&
Cancel	False
Caption	STEP
Default	False
DisabledPicture	(None)
DownPicture	(None)

(Name)
Returns the name used in code to identify an object.

Make sure you set both the Name and Caption properties to STEP.

Scripting the Button's Action

Double-click the button to open its code window. Enter a SMARTSCRIPT command into the button like this:



```
Option Explicit

Private Sub STEP_Click()
SMART.CueLocateNumber "P1", 3
End Su
```

CueLocateNumber(*device*, ***numericPosition As Long***)

You start with the name SMART, which is the name you gave to the SMART-SCRIPT icon after adding it to the form. When you type the period following the word SMART, a list of the possible actions that can be performed by SMART-SCRIPT appears. Choose CueLocateNumber from this list, then type a space. This displays the additional parameters required for this action as a small window just below the script you're entering. Enter "P1" (including the quotes) for the device parameter. This is the name of the device that is to perform the cue. Now type a comma, followed by the second parameter, which is the numeric position to locate. Enter the digit 3 here.

◆ **NOTE:** For the above to work, it is assumed that the system configuration indeed includes a device named “P1” that is capable of locating to a numeric position. If that's not the case, SMARTSCRIPT will display an error message when you attempt to run the script.

Running Your Script

To try out your script, choose "Start" on the Run menu (or press F5). Click the STEP button to tell the device "P1" to locate position 1. In order to actually see something happening, you must of course have the physical device connected, as described under "Hardware Hook-up" on page 19.

If you have another system configuration, then substitute an appropriate cue, device name and other parameters in the above example. If you attempt to use a kind of cue on a device to which it can not apply, or if you refer to a device that doesn't exist in the current system configuration, an error message will be displayed when you attempt to run the script (see "Errors" on page 47).

Where to Go from Here

Read the reference chapter, page 32, to learn more about the various cue types and other scripting commands and properties. Open up the source code for the enclosed example project, and look through the forms and their code. If you're new to Visual Basic, you may also want to read one of the many Visual Basic books available in most bookstores.

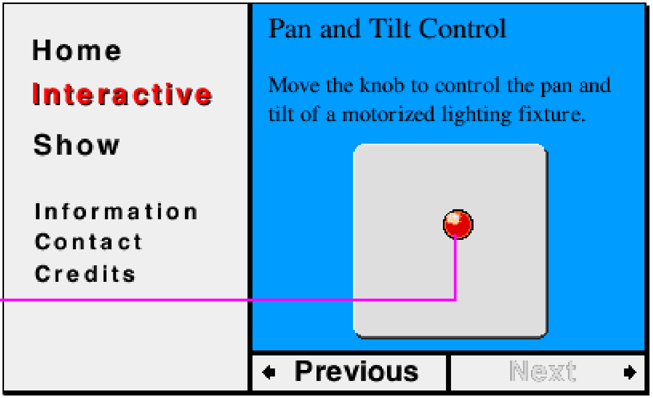
3 EXAMPLES

This chapter provides some examples of what SMARTSCRIPT applications may look like and accomplish. The screen-shots are taken from the sample applications that are enclosed with SMARTSCRIPT.

Macromedia Director

The example below shows a simple pan and tilt control for a motorized lighting fixture. The X and Y position of the lamp is controlled simply my moving the red knob. This is an example of direct, interactive device control. The "Show" section of the same sample application demonstrates how devices can be orchestrated together with the graphical elements using Director's Score window. By controlling both the on-screen, graphical elements and the external devices from the Score window, good synchronization can be achieved.

This knob moves the spotlight.

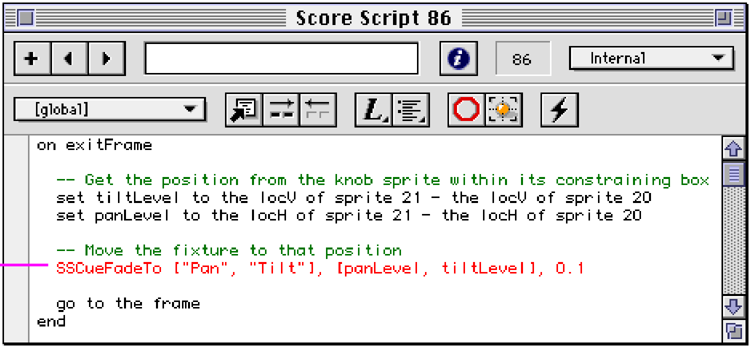


Pan and Tilt Control

Move the knob to control the pan and tilt of a motorized lighting fixture.

The Director programming required to handle the interactive pan and tilt function is surprisingly simple. As you can see below, only a single line of code is needed to control the device (shown in red). The remaining Lingo code is required for calculating the pan and tilt values based on position of the knob. The knob is managed by sprite number 21. Sprite 20 is a box that constrains the knob's movement. The pan and tilt positions are calculated by taking the difference between the horizontal and vertical positions of the knob and its enclosing box.

This single SMARTSCRIPT command controls the spotlight's position.



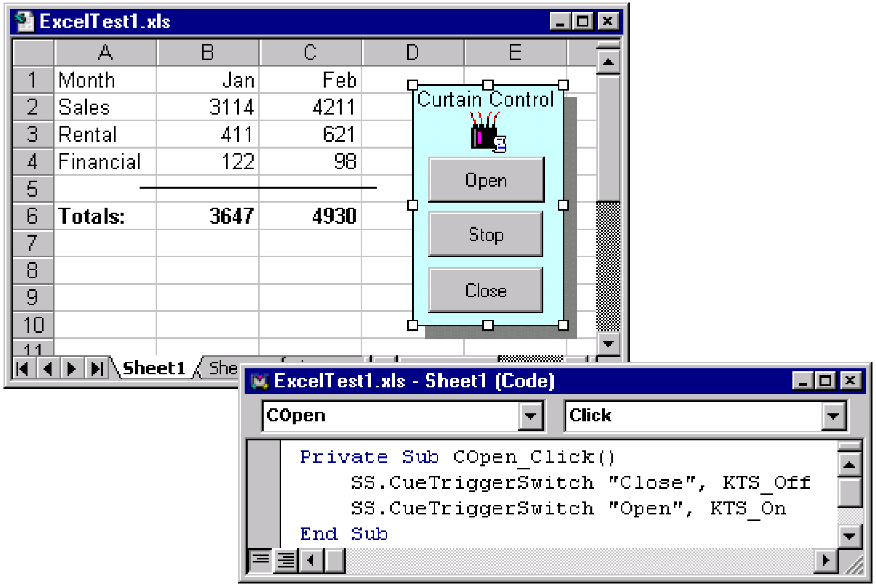
```
on exitFrame

  -- Get the position from the knob sprite within its constraining box
  set tiltLevel to the locV of sprite 21 - the locV of sprite 20
  set panLevel to the locH of sprite 21 - the locH of sprite 20

  -- Move the fixture to that position
  SSCueFadeTo ["Pan", "Tilt"], [panLevel, tiltLevel], 0.1

  go to the frame
end
```

The resulting pan and tilt levels are then used as parameters to the SSCueFadeTo SMARTSCRIPT command. Note how they are passed to SSCueFadeTo as an array by putting them within square brackets. Likewise, the names of the "Pan" and "Tilt" functions to be controlled are also passed as a corresponding array (see "Specifying Multiple Devices" on page 34).

The Macromedia Director sample application includes numerous other examples – both interactive and Score-controlled. Use the menu to the left and the Next/Previous buttons in the lower right corner to see all the examples.

Microsoft Excel

The example below shows how a few buttons have been added to an Excel spreadsheet in order to control a curtain. Note the SMARTSCRIPT icon, providing the control functions, located just above the three buttons. (In a normal application, the SMARTSCRIPT icon would of course be hidden.)



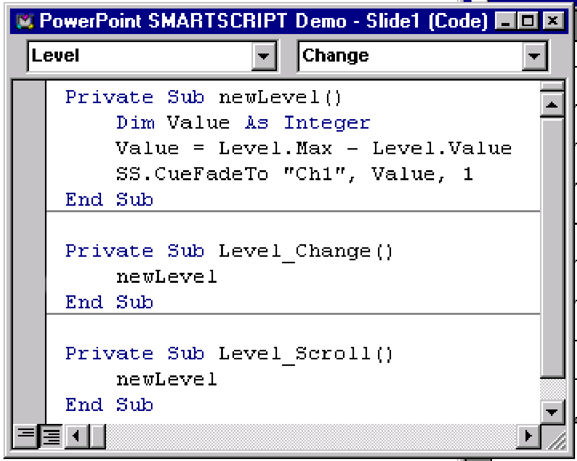
	A	B	C	D	E
1	Month	Jan	Feb		
2	Sales	3114	4211		
3	Rental	411	621		
4	Financial	122	98		
5					
6	**Totals:**	**3647**	**4930**		

```
Private Sub COpen_Click()
    SS.CueTriggerSwitch "Close", KTS_Off
    SS.CueTriggerSwitch "Open", KTS_On
End Sub
```

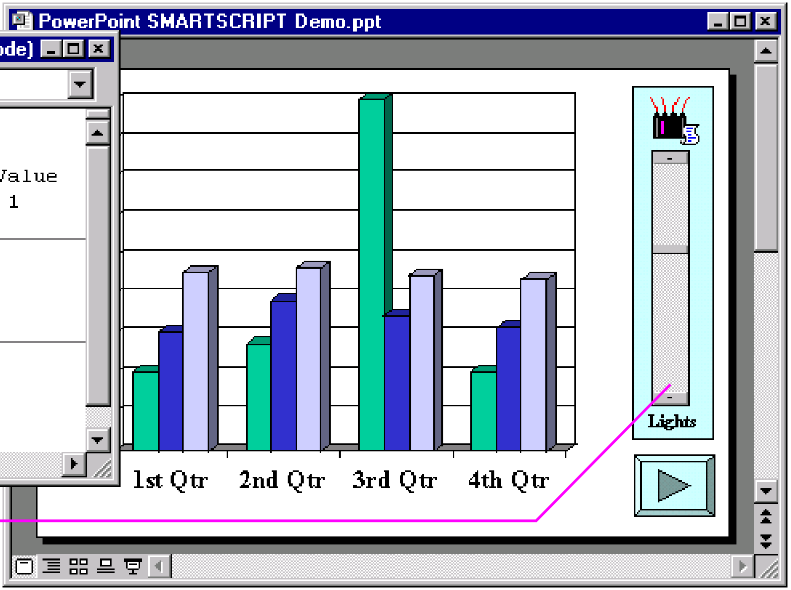
The script shown above is displayed by double-clicking the Open button. It contains two SMARTSCRIPT commands – one to turn off the "Close" switch and another to turn on the "Open" switch. This ensures that the "Close" switch always is turned off before activating the "Open" switch.

Microsoft PowerPoint

The example below shows how SMARTSCRIPT and a slider has been added to a PowerPoint slide to control the room lights.



```
Private Sub newLevel()
    Dim Value As Integer
    Value = Level.Max - Level.Value
    SS.CueFadeTo "Ch1", Value, 1
End Sub

Private Sub Level_Change()
    newLevel
End Sub

Private Sub Level_Scroll()
    newLevel
End Sub
```



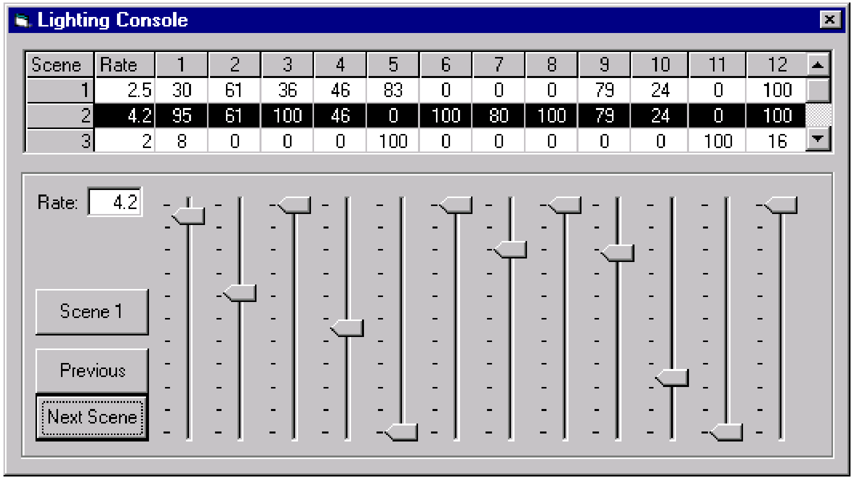
Move this slider to dim the roomlights.

The script shown above to the left is displayed by double-clicking the slider. It needs two handlers in order to respond both to the up/down arrows at the ends of the slider as well as the slider's knob. Both these handlers call on a common sub-handler – named newLevel – which calculates the new value and uses a CueFadeTo command to set the lamp named "Ch1" to the new percentage value.

Microsoft Visual Basic

The example below shows a simple, 12-channel "lighting console". Each lighting scene you create using the sliders is stored in the spreadsheet-like grid at the top.

In addition to the level of each channel, you can also specify the fade rate for each scene. This fade rate is used when the scene is activated, either by clicking it in the list or using the Next/Previous buttons.



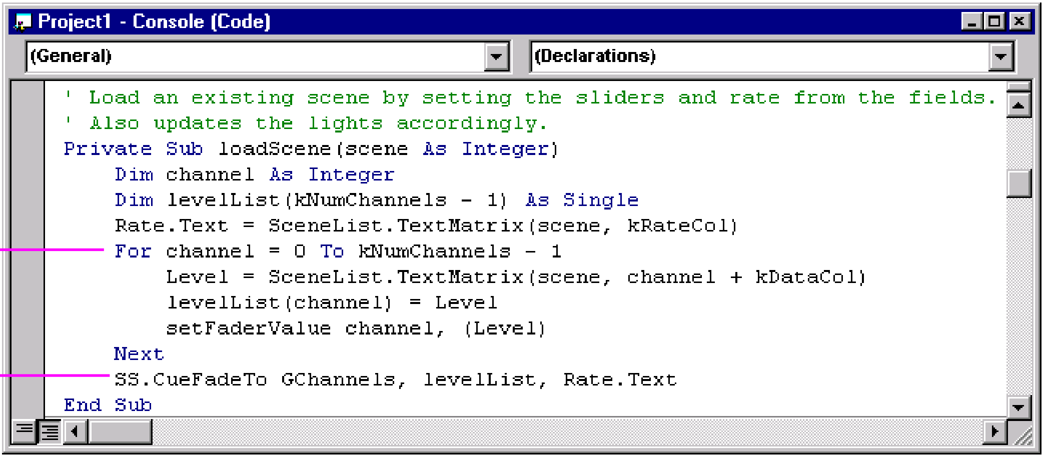
Scene	Rate	1	2	3	4	5	6	7	8	9	10	11	12
1	2.5	30	61	36	46	83	0	0	0	79	24	0	100
2	4.2	95	61	100	46	0	100	80	100	79	24	0	100
3	2	8	0	0	0	100	0	0	0	0	0	100	16

The sliders, buttons and the grid at the top are all standard items that come as part of Visual Basic. Some controls are built into Visual Basic and appear in the toolbox automatically. Other controls must be activated manually just like SMARTSCRIPT (see "Adding SMARTSCRIPT to the Toolbox" on page 20).

The visual basic script shown below is performed when you go to a scene using the Next/Previous buttons (see previous page), or by clicking a scene in the list. The desired scene number is provided as a parameter to the loadScene subroutine.

The For loop collects the scene levels and sets the faders accordingly.

The CueFadeTo command fades all the channels to the specified levels.



```
' Load an existing scene by setting the sliders and rate from the fields.
' Also updates the lights accordingly.
Private Sub loadScene(scene As Integer)
    Dim channel As Integer
    Dim levelList(kNumChannels - 1) As Single
    Rate.Text = SceneList.TextMatrix(scene, kRateCol)
    For channel = 0 To kNumChannels - 1
        Level = SceneList.TextMatrix(scene, channel + kDataCol)
        levelList(channel) = Level
        setFaderValue channel, (Level)
    Next
    SS.CueFadeTo GChannels, levelList, Rate.Text
End Sub
```

As you can see from the script, the amount of coding required to support SMARTSCRIPT is very modest. Most of the code in a typical SMARTSCRIPT-based application deals with the user interface – ie, windows, buttons and sliders – and other details related to Visual Basic itself.

4 REFERENCE

This chapter looks at SMARTSCRIPT commands, properties and constants.

Macromedia Details

This section describes details pertaining to the Xtra implementation of SMARTSCRIPT for Directo,r running under MacOS or Windows.

Accessing SMARTSCRIPT

Assuming that you have installed SMARTSCRIPT as described under “Installation” on page 10, all SMARTSCRIPT commands are directly available from Lingo. If you want to move SMARTSCRIPT to another computer, all you need is the SMARTSCRIPT Xtra (“SMARTSCRIPT.Mac” on MacOS, SMARTSCRIPT.x32 on Windows), which must be placed in the Xtra folder, located in the same folder as your Director application.

Before you can perform any of the Cue functions, you must first start the communication using the Open command (see “Open” on page 49).

Calling Conventions

When performing a cue or other SMARTSCRIPT command, you simply append any parameters right after the command. Parameters are comma separated.

```
SSCueTriggerSwitch "Open", #KTS_On
```

The command above takes two parameters; the reference to the switch device to be controlled and what to do with it (ie, turning it on). There’s a blank between the command and its first parameters.

◆ **IMPORTANT:** In the Xtra implementation of SMARTSCRIPT, all command and function names are preceded by “SS”, as shown above. There’s no space or other character between the “SS” and the command’s name.

Functions

In addition to commands, SMARTSCRIPT also provides some functions. The main difference between a command and a function is that a function returns a result to the Lingo script. This result can be stored in a variable, or used in calculations and other expressions.

This is an example of how you can convert a timecode position from a string representation to its numeric form:

```
set numericTime = SSConvertStringToTime("3:22/13", #KTF_SMPTE_DropFrame)
```

There are two main differences between a function, as shown above, and regular commands:

- Parameters to functions are enclosed within parentheses.
- The value returned from the function must be used in an expression or stored in a variable.

The function shown above converts the string "3:22/13" from SMPTE drop-frame format to a number. This is often useful if you need to make calculations involving time values. The result is stored in a variable named numericTime.

◆ **IMPORTANT:** When calling a function from Lingo, you must always include the parentheses after the function name. This applies even if the function doesn't take any parameters, in which case there won't be anything at all between the parentheses.

Constants

Some SMARTSCRIPT commands use predefined names to specify certain parameters. These predefined names are called constants. For example, the last parameter shown below is a constant defined by SMARTSCRIPT:

```
set numericTime = SSConvertStringToTime("3:22/13", #KTF_SMPTE_DropFrame)
```

The KTF_SMPTE_DropFrame constant specifies what kind of timecode to convert from.

◆ **IMPORTANT:** In Lingo, constant names must be preceded by the #-sign, as in the example above. There's no space between the #-sign and the constant name.

Specifying Multiple Devices

Sometimes you may want to apply a cue to more than one device. You can do so by passing a list of device references, enclosed within square brackets:

```
SSCueDissolve ["P1", "P2"], 2.5, True
```

This cue performs a 2.5 second dissolve on device "P1" and "P2". The list can contain either device names or device indexes (see "Device Parameter" on page 49).

Errors

When an error occurs in SMARTSCRIPT, it displays a message in the Message window telling you what went wrong. Unfortunately, it is not possible for SMARTSCRIPT to stop running your Lingo script and highlight the offending statement.

These error messages are useful while creating and debugging your application. However, you may want to turn off any error messages originating from Cue commands once you've finished testing your application. If IgnoreErrors is set to TRUE, any errors in Cue commands will not be reported in the Message window (see “IgnoreErrors” on page 68).

Regardless of whether IgnoreErrors is set to TRUE or FALSE, you can always retrieve the error code related to the most recent SMARTSCRIPT command or function using the LastError function (see “LastError” on page 69).

Windows Details

This section describes programming details pertaining specifically to the ActiveX implementation of SMARTSCRIPT for Windows.

◆ **NOTE:** If you’re using the Xtra implementation of SMARTSCRIPT under Windows, please refer to “Macromedia Details” on page 32 for details.

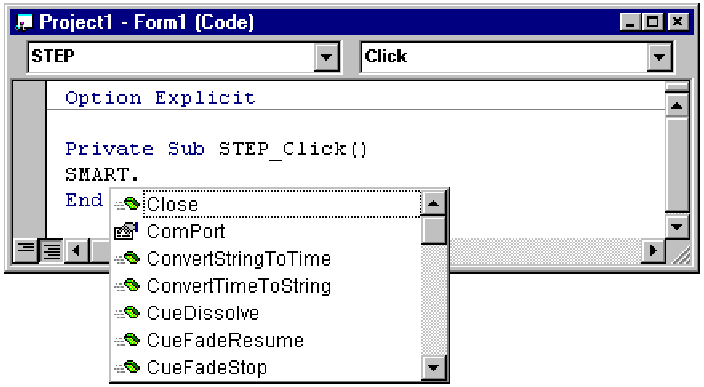
Accessing SMARTSCRIPT

In order to perform one of the SMARTSCRIPT commands or functions, you must first draw a SMARTSCRIPT icon in a window (called “form” in Visual Basic). See “Adding SMARTSCRIPT to the Main Form” on page 22. There can only be a single SMARTSCRIPT icon active at a time.

◆ **NOTE:** The SMARTSCRIPT icon will not be drawn when you run your finished application – it is only visible during design.

After adding the SMARTSCRIPT icon to the window, you must name it by setting its Name property. In Visual Basic, this is done using the Properties window. Make sure that the SMARTSCRIPT icon remains selected in the form window. Use a short name, such as SMART or SS, as you’ll need to use this name whenever you want to perform any SMARTSCRIPT command.

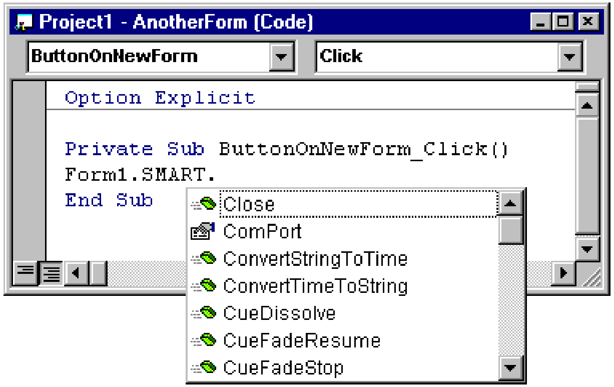
In Visual Basic, the SMARTSCRIPT object becomes accessible to all handlers in that form using its assigned name. Thus, assuming that you set the Name property of the SMARTSCRIPT object to "SMART", you can then add a button to the same form that accesses the SMART object. After drawing a button, double-click it to open its handler and enter SMART followed by a period:



```
Option Explicit

Private Sub STEP_Click()
SMART.
End
```

Accessing SMARTSCRIPT from Other Forms

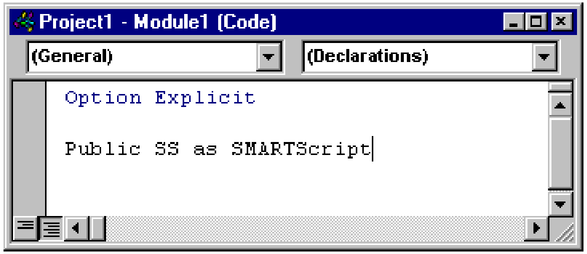
To access SMARTSCRIPT from handlers in other forms (eg, using a button in another window), you must qualify the SMART object by prefixing it with the name of the form containing the SMARTSCRIPT icon. Assuming that the SMARTSCRIPT icon sits on a form named “Form1”, you can access it from another form by typing the name of the form, a period, and then the name of the SMARTSCRIPT object, followed by another period.



```
Option Explicit

Private Sub ButtonOnNewForm_Click()
Form1.SMART.
End Sub
```

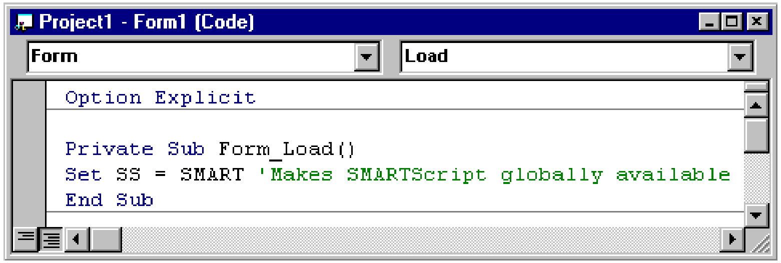
Accessing SMARTSCRIPT from All Form

Alternatively, you can make SMARTSCRIPT available to all handlers on all forms by publishing it using a global variable. Once this is done, you can refer to the SMARTSCRIPT object using the name of this global variable. In essence, the variable becomes a universally accessible alias to the SMARTSCRIPT icon. In Visual Basic, you must establish this global variable in a separate code module – it can not be established from within a form's code module. To create such a separate code module, choose "Add Module" on the Project menu, then add the line shown below to that module.



```
Option Explicit

Public SS as SMARTScript
```

After having established this global variable, you must link it to the SMART-SCRIPT icon you placed on the form. To do this, use a Set statement in the Load handler of the form containing the SMARTSCRIPT icon. Double-click the background of the form containing the SMARTSCRIPT icon, and add the Load handler as shown below:



```
Option Explicit

Private Sub Form_Load()
Set SS = SMART 'Makes SMARTScript globally available
End Sub
```

Once this is done, you can access SMARTSCRIPT from any handler through the name of the global variable, in this case SS, followed by a period.

◆ **NOTE:** You can't use the same name for the global variable and for the SMARTSCRIPT icon. For the sake of consistency, you may prefer to access SMARTSCRIPT through the global variable's name even from handlers on the main form. If not, code you copy from the main form to other forms will no longer work, since the name is then no longer accessible.

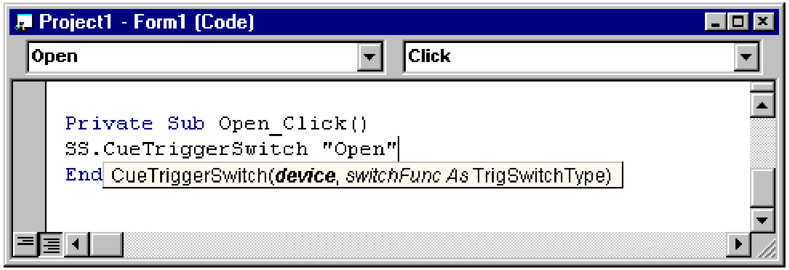
Calling Conventions

When performing a cue or other SMARTSCRIPT command, you append any parameters to the command. Parameters are comma separated:

```
SS.CueTriggerSwitch "Open", KTS_On
```

In the above example, SS is the name of the SMARTSCRIPT global variable, and CueTriggerSwitch is the name of the command to be performed. This command takes two parameters; the reference to the switch device to be controlled and what to do with it (ie, turning it on). There's a blank between the command and its first parameters. Subsequent parameters are separated by a comma.

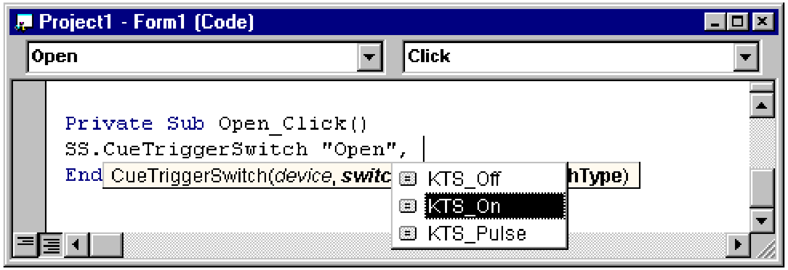
When typing the period following the SMART component name or SS variable name, Visual Basic displays a list of SMARTSCRIPT's commands and properties. Choose the desired item from this list, or type it on the keyboard. Press the space bar to complete the command or property name. If the command requires any additional parameters, a template for the expected parameters will be displayed below the command.



```
Private Sub Open_Click()
SS.CueTriggerSwitch "Open"
End
```

CueTriggerSwitch(**device**, *switchFunc As* TrigSwitchType)

If a parameter can be specified using a constant, a list of the appropriate constants will appear when you reach that parameter.



```
Private Sub Open_Click()
SS.CueTriggerSwitch "Open", |
End
```

Functions

In addition to commands, SMARTSCRIPT also provides some functions. The main difference between a command (called a “Sub” in Visual Basic) and a function is that a function returns a result to the script. This result can be stored in a variable, or used in calculations and other expressions.

This is an example of how you can convert a timecode position from string representation to numeric form:

```
numericTime = SS.ConvertStringToTime(“3:22/13”, KTF_SMPTE_DropFrame)
```

There are two main differences between a function, as shown above, and regular commands:

- Parameters to functions are enclosed within parentheses.
- The value returned from the function must be used in an expression or stored in a variable.

The function shown above converts the string “3:22/13” from SMPTE drop-frame format to a number. This is often useful if you need to make calculations involving time values. The result is stored in the variable named numericTime.

Properties

Certain aspects of SMARTSCRIPT can be controlled through properties. Some properties simply provide alternative methods to perform some functions. Other properties address ActiveX-specific details.

An advantage of properties is that they can be configured manually using the Properties window. This allows you to specify which serial port to use and the name of the system configuration file, without writing any program code (see “Configuring SMARTSCRIPT” on page 22).

However, properties can also be accessed from your program. This statement sets the IgnoreErrors property to True:

```
SS.IgnoreErrors = True
```

◆ **NOTE:** When you stop running your Visual Basic program, properties revert to their initial settings, as specified manually in the Properties window. Thus, if you change a property in your program and then stop the program, the change won’t appear in the Properties window. To see the value of a property while running, use the debugging capabilities built into your development environment.

You can read or test the current value of a property simply by referring to it. This is similar to how you access SMARTSCRIPT functions, but without the function parameters. The Visual Basic example below displays a message box if the IgnoreErrors property is set to True:

```
if SS.IgnoreErrors then MsgBox(“Warning: Error detection is currently disabled!”)
```

This is a list of all the properties in the ActiveX implementation of SMART-SCRIPT:

Property	*Description*
Version	Returns the version number of SMARTSCRIPT, as a three-digit integer. The value 100 means version 1.0.0.
IgnoreErrors	Determines whether any runtime errors originating from Cue commands should be ignored or reported. Possible values are False and True.
ComPort	Specifies which serial port to use to communicate with SMARTPAX. Possible values are KCP_Unspecified, KCP_COM1, KCP_COM2, KCP_COM3 and KCP_COM4.
SysDescFile	Specifies the drive, path and file-name of the system description file, as a string.

Version. This property is read-only, and therefore doesn't appear in the Property window. It can only be accessed from your program.

IgnoreErrors. Set this property to True to ignore runtime errors originating from cues. If set to False, runtime errors will be reported, and may interrupt your application by displaying an error message. Typically, you have this property set to False during development and testing, and set it to True when you compile the final application. See "Handling Errors" on page 71 for more details.

ComPort. This property is provided as an alternative to the Open command to specify which serial port to use. If you specify the port using this property, you must also specify the name of the system description file using the SysDesc-File property.

SysDescFile. This property is provided as an alternative to the Open command for specifying the name of the system description file (see "System Description File" on page 14). If you specify the filename using this method, you must also specify which serial port to use using the ComPort property.

◆ **NOTE**: Use the ComPort and SysDescFile properties in preference to the Open command. It is it easier to change these settings, and ensures that the description file and serial port are opened without the need for any additional coding on your behalf. Furthermore, by using the properties instead of the Open command, any errors will be reported at design time. If you use the Open command, errors won't be reported until runtime.

Additional Properties. The Property window in Visual Basic lists some additional properties, such as Index and Tag. These are not used by SMART-SCRIPT, and you should leave them blank. The Name property (displayed within parentheses in the Property window) gives SMARTSCRIPT its name, which is used to refer to it in your program. Thus, if you change this property, any scripts you have written using the old name will fail.

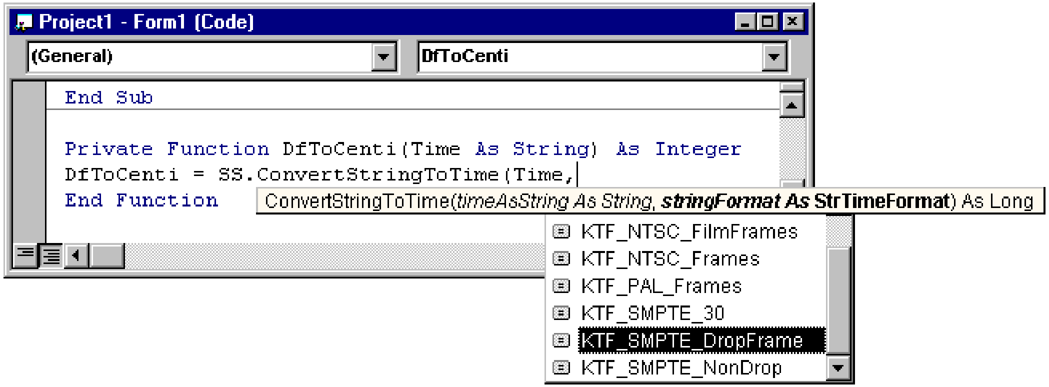
Constants

Some SMARTSCRIPT commands use predefined names to specify certain parameters. These predefined names are called constants. For example, the last parameter shown below is a constant defined by SMARTSCRIPT:

numericTime = SS.ConvertStringToTime("3:22/13", KTF_SMPTE_DropFrame)

The KTF_SMPTE_DropFrame constant specifies from which kind of timecode to convert. Note that constants are not strings, and therefore are not enclosed within quotes.

When you reach a parameter that expects a constant, Visual Basic displays a list of the applicable constants.



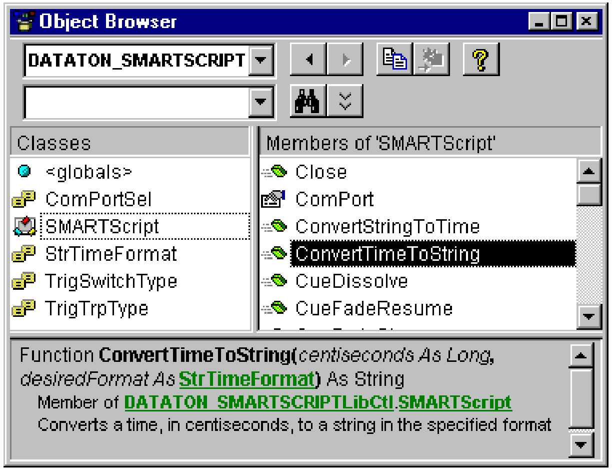
```
End Sub

Private Function DfToCenti(Time As String) As Integer
DfToCenti = SS.ConvertStringToTime(Time,
End Function
```

ConvertStringToTime(*timeAsString As String*, ***stringFormat As* StrTimeFormat**) As Long

- KTF_NTSC_FilmFrames
- KTF_NTSC_Frames
- KTF_PAL_Frames
- KTF_SMPTE_30
- KTF_SMPTE_DropFrame
- KTF_SMPTE_NonDrop

Using the Object Browser

To see a complete list of all available commands, functions, properties and constants, use the "Object Browser" command on the View menu in Visual Basic. Select DATATON_SMARTSCRIPT on the Library menu, click the desired class in the Class list, and choose the item in the Members list. This displays information about the selected item in the field at the bottom of the Object Browser window.



Function **ConvertTimeToString**(*centiseconds As Long, desiredFormat As* StrTimeFormat) As String
Member of DATATON_SMARTSCRIPTLibCtl.SMARTScript
Converts a time, in centiseconds, to a string in the specified format

Specifying Multiple Devices

Sometimes you may want to apply a cue to more than one device. Do so by passing an array of device references as the *device* parameter to the command:

```
SS.CueDissolve Array("P1", "P2"), 2.5, True
```

This cue performs a 2.5 second dissolve on device P1 and P2. Notice how the Array function, built into Visual Basic, is used to create the array on the fly. The array can contain either device names or device indexes (see “Device Parameter” on page 49).

Errors

When an error occurs in SMARTSCRIPT, this information is passed back to the host application, which typically displays an error message. The error message includes the error code, which describes the cause for the error (see “Error Codes” on page 70).

While such error messages are very useful in developing and debugging your application, they’re generally not desired in the finished software. There are two methods by which you can avoid such error messages; setting the IgnoreErrors property to True or handling errors explicitly.

Once you have thoroughly debugged and tested your application, you normally set the IgnoreErrors property to True before making the final application. This will prevent most errors originating from CueXxx commands and similar from being reported. In this case, when an error occurs, the command will simply be ignored, and your application will continue to run as if no error had occurred.

The other possibility is to use the error trapping capabilities built into the host development environment. In Visual Basic, for example, you can use the “On error” statement to control what to do in case of an error. This method is particularly useful when dealing with user input, where you may prefer to beep or display a message if the user enters an invalid value. The “Timecode Calculator” form in the Visual Basic sample application demonstrates how this is used to handle errors originating from the ConvertStringToTime function.

Data Types

In the syntax description of each command, function and property throughout the remainder of this chapter you will find references to data types. Each parameter to a function has a particular type, such as string or integer.

These are the types used in SMARTSCRIPT:

Data type	*Description*
string	A character string, enclosed in double quotes. Used to specify device names and some other parameters.
integer	A whole number, such as 352, -23 or 0.
float	A number that accepts an optional fractional part, such as 3.97.
boolean	A true/false value, specified using the corresponding keywords of the host development language.
variant	Indicates a parameter whose type may vary depending on the circumstances. Also used for arrays.
constant	A named identifier from a list of predefined constants provided by SMARTSCRIPT, such as the StrTimeFormat constants shown on page 66.

Device Parameter

The device parameter, used with all Cue commands, is defined as a variant type. It can be any of the following three:

- A string, specifying the name of the device, such as "LDP".
- An integer, specifying the index number of the device (see "DevName-ToDevIndex" on page 64).
- An array, containing either of the above. This allows you to specify multiple devices to be affected by the same cue. In Visual Basic, you create an array using the Array function. In Lingo, you simply enclose the array elements within square brackets. See "Specifying Multiple Devices" on page 34 and page 46 for examples.

Establishing Communication

This section describes how to establish and tear down the communication through SMARTSCRIPT. When using the Director Xtra implementation of SMARTSCRIPT, you must explicitly open and close the communication. In the Windows ActiveX implementation, you may use either the explicit Open and Close commands, or use the ComPort and SysDescFile properties. If you use the properties then you don't need to call Open and Close (this is the recommended method).

Open

Opens the communications path through SMARTSCRIPT to the SMARTPAX control units and the devices. You must specify the name of a serial port and the name of the system description file. Unless the system description file is in the same directory as your application, you must specify its full path name.

Syntax:

```
Open string configFileName, string serialPortName
```

ActiveX example:

```
SS.Open "C:\Dataton\MySystem.SDF", "COM1"
```

Opens SMARTSCRIPT using the COM1 serial port and the system description file named "MySystem.SDF", located in the Dataton subdirectory on the C: drive.

◆ **NOTE:** Do not use Open if you're using the ComPort and SysDescFile properties in the ActiveX implementation.

Xtra example: `SSOpen "HD:Dataton:MySystem.SDF", "Modem"`

Opens SMARTSCRIPT using the Macintosh Modem port and the system description file named "MySystem.SDF", located in the Dataton sub-folder on the disk named "HD".

◆ **NOTE:** When specifying a file path on MacOS, folder and file names are separated by colon. Windows uses a backslash for this purpose.

Alternatively, you can store the system description file in the same folder as the movie file. Then use the built-in Lingo function "the moviePath" to access the file, as shown in the example on page 16. This avoids putting a hard-coded path specification in the script, thus making it more portable.

In order to make your SMARTSCRIPT application portable across platforms, SMARTSCRIPT will silently convert between the following serial port names on the Mac and the PC:

Macintosh serial port name	*PC serial port name*
Modem	COM1
Printer	COM2

Close

Closes SMARTSCRIPT, freeing the serial port and any other system resources it may have been using.

Syntax: Close

ActiveX example: SS.Close

Do not use Close if you're using the ComPort and SysDescFile properties in the ActiveX implementation.

Xtra example: SSClose

Version

Returns the version number of SMARTSCRIPT, as an integer. The value 100 means version 1.0.0. You can use this to check which version number of SMARTSCRIPT is being used, in case you depend on features made available in a particular version.

Syntax: integer Version

ActiveX example: if SS.Version < 101 then MsgBox "SMARTSCRIPT is too old, update to the latest version"

Xtra example: if SSVersion() < 101 then Alert "SMARTSCRIPT is too old, update to the latest version"

◆ **NOTE:** In the ActiveX implementation, Version is implemented as a read-only property. In the Xtra implementation it's a function, as indicated by the empty parentheses in the example above.

ComPort

Specifies which communications port to be used. This property is available only in the ActiveX implementation of SMARTSCRIPT. In the Xtra implementation you use the Open command to achieve the same result.

This property can also be set through the Properties window. If you specify the ComPort property, you must also specify the SysDescFile property. Do not use the Open command if you use these properties.

Syntax:

```
ComPortSel ComPort
```

where ComPortSel is one of the following constants:

ComPortSel	*Description*
KCP_Unspecified	Port not yet specified, or closed.
KCP_COM1	Serial port COM1 being used.
KCP_COM2	Serial port COM2 being used.
KCP_COM3	Serial port COM3 being used.
KCP_COM4	Serial port COM4 being used.

ActiveX example:

```
SS.ComPort = KCP_COM2
```

Selects COM2 as the serial port to use to talk to the SMARTPAX units. Use one of the predefined ComPortSel constants to specify this property.

SysDescFile

Specifies the name of the system description file (see "System Description File" on page 14). This property is available only in the ActiveX implementation of SMARTSCRIPT. In the Xtra implementation you use the Open command to achieve the same result.

This property can also be set through the Properties window. If you specify the SysDescFile property, you must also specify the ComPort property. Do not use the Open command if you use these properties.

Syntax: string SysDescFile

ActiveX example: SS.SysDescFile = "C:\Dataton\MySystem.SDF"

Opens the file named MySystem.SDF, located in the Dataton directory on the C: drive.

Performing Cues

The Cue commands control the devices connected through the SMARTPAX units. These commands are modelled after the cues available in TRAX. Thus, if you're familiar with TRAX, you will recognize all the cues and their parameters. If you're unfamiliar with TRAX, you should read chapter 7 in the TRAX 3 manual (available free of charge at www.dataton.com, under the "free software" heading).

All cues take as their first parameter a reference to the device or devices to be controlled. You can refer to a device either using its name or its index number (see "Device Parameter" on page 49). If you want to apply a cue to multiple devices, you can specify an array of device references (see "Specifying Multiple Devices" on page 34 and page 46).

All ActiveX examples below assume that you have a global variable named SS that refers to the SMARTSCRIPT component (see "Accessing SMARTSCRIPT from All Form" on page 38). Alternatively, you can name the SMARTSCRIPT component itself SS, but then the examples will only work in handlers located in the same form as the SMARTSCRIPT icon itself.

CueLocateNumber

Locates a discrete, numeric position, such as a slide in a slide projector or a song on a CD.

Syntax: `CueLocateNumber variant device, integer numericPosition`

ActiveX example: `SS.CueLocateNumber “Director”, 35`

Locates position 35 in the device named “Director”.

Xtra example: `SSCueLocateNumber 2, 0`

Locates position 0 of the device with index 2.

CueLocateNumberRelative

Locates a numeric position relative the current position.

Syntax: `CueLocateNumberRelative variant device, integer numberOfSteps`

ActiveX example: `SS.CueLocateNumber "Denon", -1`

Locates the previous song on the device named “Denon”

Xtra example: `SSCueLocateNumber "Denon", -1`

CueLocateTime

Locates a time position on a device having a continuous-time medium, such as a tape or disc, that can be located to an arbitrary position or frame.

Syntax: `CueLocateTime variant device, integer inTimeInCentiseconds, integer outTimeInCentiseconds`

The *inTimeInCentiseconds* parameter specifies the time position to locate, in centiseconds. Use the ConvertStringToTime function to specify the time using other standard timecode and frame-code formats (see “Time Formats” on page 65).

The *outTimeInCentiseconds* parameter specifies the target position where the subsequent CueTriggerTransport play cue will stop automatically. This feature is not supported by all devices. If you don't want to specify an out point, then set *outTimeInCentiseconds* to zero, as shown in the examples below.

ActiveX example:

```
SS.CueLocateTime "LDP", SS.ConvertStringToTime("10:00", KTF_EBU_25), 0
```

Locates the time position 10 minutes on the device named "LDP". This example uses the ConvertStringToTime function (see page 67) to convert from a human readable time, as a string, to the standard time used in SMARTSCRIPT, which is centiseconds.

Xtra example:

```
SSCueLocateTime "Beta", 350, 0
```

Locates the time position 350 on the device named "Beta". The time specifies centiseconds (ie, 350 equals 3 seconds and 50 hundredths). Use the Convert-StringToTime function to specify the time using any other desired format.

CueSetTo

Sets the output level to a percentage value. Controls audio, video, lighting, etc.

◆ **IMPORTANT:** For some devices, you must use this command to set the output level to 100 before you will see or hear anything while playing. This includes many video disc players as well as some audio devices.

Syntax:

```
CueSetTo variant device, float levelInPercent
```

ActiveX example:

```
SS.CueSetTo "LDP", 100
```

Turns on the video output of the laserdisc player named "LDP".

Xtra example:

```
SSCueSetTo "Mic3", 65
```

Sets the level of microphone "Mic1" to 65 percent.

CueFadeTo

Fades the output level to specified percentage with the specified rate. This is similar to CueSetTo, but allows you to fade the level gradually instead of setting it instantly. It also allows you to fade multiple devices to individual levels.

If you specify an array of devices in the *device* parameter, you can specify a corresponding array of target levels in the *levelInPercent* parameter. If you do, each device will be faded to the corresponding level, allowing you to set a complete lighting scene with one command, where each channel can have its own value. See the Console form in the enclosed Visual Basic sample application for an example of how this can be used.

Syntax:

```
CueFadeTo variant device, variant levelInPercent, float timeInSeconds
```

If *device* is an array, *levelInPercent* may be a corresponding array of levels. If *levelInPercent* isn't an array, or contains fewer elements than the *device* array, the last (or only) value in the *levelInPercent* parameter will be applied to any remaining devices in the *device* array. If *levelInPercent* contains more values than *device*, the extraneous values will be ignored.

ActiveX example:

```
SS.CueFadeTo Array("Ch1", "Ch2", "Ch3"), Array(75, 12, 20), 3.5
```

Fades the specified three lighting channels to the corresponding percentages over 3.5 seconds.

Xtra example:

```
SSCueFadeTo "Mic3", 0, 2
```

Fades out the sound channel “Mic3” using a two second fade rate.

To use arrays in the Xtra implementation (called “lists” in Lingo), simply enclose the array elements in square brackets:

```
SSCueFadeTo ["Ch1", "Ch2", "Ch3"], [75, 12, 20], 3.5
```

CueFadeStop

Stops a fading or dissolve initiated by a CueFadeTo or CueDissolve command. This can be used to stop a fading at its current level, for example to implement "Up", "Down" and "Stop" buttons.

Syntax:	CueFadeStop variant device
ActiveX example:	SS.CueFadeStop "Ch3"
	Stops fading the light level of "Ch3" at its current level.
Xtra example:	SSCueFadeStop "Mic3"

CueFadeResume

Resumes a fading stopped using the CueFadeStop command. The fading resumes in the same direction and with the same rate as before it was stopped.

Syntax:	CueFadeResume variant device
ActiveX example:	SS.CueFadeResume"Ch3"
Xtra example:	SSCueFadeResume "Mic3"

CueTriggerSwitch

Controls the output of a switch device (eg, a relay).

Syntax:

```
CueTriggerSwitch variant device, TrigSwitchType whatToDo
```

Where *whatToDo* is one of the following predefined constants:

TrigSwitchType	*Description*
KTS_On	Activates the switch
KTS_Off	Releases the switch.
KTS_Pulse	Pulses the switch for 0.2 second.

ActiveX example:

```
SS.CueTriggerSwitch "ScrUp", KTS_On
```

Activates the switch named “ScrUp”.

Xtra example:

```
SSCueTriggerSwitch "Open", #KTS_Off
```

Releases the switch named “Open”.

◆ **NOTE:** In the Xtra implementation of SMARTSCRIPT, constants must be prefixed with a #-sign.

CueTriggerTransport

Controls transport functions of players, such as audio and video disc devices and tape devices.

Syntax:

```
CueTriggerTransport variant device, TrigTrpType whatToDo
```

Where whatToDo is one of the predefined constants:

TrigTrpType	*Description*
KTT_Stop	Stops the transport.
KTT_Pause	Pauses the transport.
KTT_PlayFwd	Plays forward.
KTT_PlayRev	Plays backwards (not supported by all devices).
KTT_FastFwd	Fast forwards the device.
KTT_FastRev	Rewinds the device.
KTT_Record	Starts recording (not supported by all devices).

ActiveX example:

```
SS.CueTriggerTransport "LDP", KTT_PlayRev
```

Plays the device named “LDP” backwards.

Xtra example:

```
SSCueTriggerTransport "Denon", #KTT_Pause
```

Pauses the device named “Denon”.

◆ **NOTE:** In the Xtra implementation of SMARTSCRIPT, constants must be prefixed with a #-sign.

CueTriggerMode

Controls device-specific modes available for some devices. Which modes are available depends on the make and model of device being controlled. Some devices don't have any device specific modes.

The name of the modes and states applicable for a device are the same as those used in TRAX. Thus, if you have TRAX available, you can use it to look up the names of these features. If you don't have TRAX available, you can see these names under that device in the system description file.

Syntax: CueTriggerMode variant device, string modeName, string stateName, integer numericValue

Sets the specified mode to the specified state with the specified value. A mode acts as a multi-position switch, where the state is the position you wish to set it to. In some cases, a state may take a numeric value. If so, this is indicated by a #-sign at the end of the state name. Pass 0 here if the state doesn't take a value.

ActiveX example: SS.CueTriggerMode "LDP", "CX", "Off", 0

Sets the "CX" mode of device "LDP" to "Off". Although the value isn't used, you must still enter 0 here.

Xtra example: SSCueTriggerMode "VProj", "Contrast", "Value#", 190

Sets the value of the "Contrast" mode of device "VProj" to 190.

CueTriggerReset

Resets the device to its initial power-up state.

Syntax: CueTriggerReset variant device

ActiveX example: SS.CueTriggerReset "LDP"

Xtra example: SSCueTriggerReset "P1"

CueTriggerEject

Ejects the medium from the device.

Syntax: `CueTriggerEject variant device`

ActiveX example: `SS.CueTriggerEject "LDP"`

Xtra example: `SSCueTriggerEject "Beta"`

CueDissolve

Turns the light on or off for the specified slide projector device at the specified rate. If the light was on prior to executing this command, it will be turned off, and vice versa. If the light goes off and *toStep* is set to true then the projector will advance to the next slide position after its light has gone out. This command works only with slide projector devices.

Syntax: `CueDissolve variant device, float rateInSeconds, boolean toStep`

ActiveX example: `SS.CueDissolve Array("P1", “P2”), 3.5, True`

Dissolves between the two slide projectors. The one that fades out will advance its tray position.

Xtra example: `SSCueDissolve ["P1", “P2”], 1, FALSE`

Dissolves between the two slide projectors.

CueFlashStart, CueFlashStop

Starts and stops flashing the specified slide projector device at the specified rate and with the specified duty-cycle. These commands works only with slide projector devices.

Syntax:

```
CueFlashStart variant device, float rateInSeconds, integer dutyCycleInPercent
CueFlashStop variant device
```

ActiveX example:

```
SS.CueFlashStart "P1", 0.5, 50
```

Starts flashing the projector at an interval of 0.5 seconds, at a 50% on/off ratio.

Xtra example:

```
SSCueFlashStart "P1", 0.3, 70
```

Starts flashing the projector at an interval of 0.3 second interval and at a 70% on and 30% off ratio.

CueSnapHard, CueSnapSoft

Controls the mechanical or simulated shutter in the specified slide projector device. These commands work only with slide projector devices.

Syntax:

```
CueSnapHard variant device, boolean toClose
CueSnapSoft variant device, boolean toClose
```

ActiveX example:

```
SS.CueSnapHard "P1", True
```

Closes the mechanical shutter in projector "P1".

Xtra example:

```
SSCueSnapSoft "P1", FALSE
```

Opens the simulated shutter in projector "P1". The simulated shutter can be used to program flash effects while fading or dissolving.

Device Indexing

The easiest way to specify a device when using cues is by name. Another possibility is to specify devices using their index number. All Cue commands accept either the name or the index number of the device (or an array of either of those).

Using index numbers is sometimes more convenient if you need to present a list of devices to the user. Using indexes is also more efficient, particularly when you access many devices repeatedly (for an example of how to do this, see the Lighting Console form in the ActiveX sample program).

The following functions allow you to translate between device names and indexes.

DevCount

Returns the number of devices in the current system configuration. Device indexes range from 1 to this number, inclusive. If the system description file hasn't been opened yet, this function returns 0.

Syntax:

```
integer DevCount()
```

ActiveX example:

```
for devIndex = 1 to SS.DevCount()
    DevList.AddItem SS.DevIndexToDevName(devIndex)
next
```

Adds all device names to a dropdown menu named DevList.

Xtra example:

```
repeat with devIndex = 1 to SSDevCount()
    put SSDevIndexToDevName(devIndex)
end repeat
```

Prints all device names in the message window.

DevNameToDevIndex

Given a device name, this function returns its corresponding index number, which is always greater than 0. If the specified device doesn't exist, this function returns 0.

Syntax:

```
integer DevNameToDevIndex(string deviceName)
```

ActiveX example:

```
for ch = 1 to kNumChannels
        GChannels(ch-1) = SS.DevNameToDevIndex("Ch" & ch)
next
```

Caches the specified device indexes to avoid excessive name lookups when later using these devices. The indexes are stored in an integer array named GChannels, which can be used as the *device* parameter to the CueFadeTo command. See the Console form in the Visual Basic sample application.

DevIndexToDevName

Given a device index, this function returns the name of that device. If the specified device doesn't exist, this function returns an empty string.

Syntax:

```
string DevIndexToDevName(integer deviceIndex)
```

ActiveX example:

```
for devIndex = 1 to SS.DevCount()
        DevList.AddItem SS.DevListToDevName(devIndex)
next
```

Adds the device names to a dropdown menu named DevList.

Xtra example:

```
repeat with devIndex = 1 to SSDevCount()
        put SSDevListToDevName(devIndex)
end repeat
```

Prints all device names in the Message window.

Miscellaneous

This group contains some miscellaneous commands, functions and properties not directly related to controlling devices. This includes format conversion and error management.

ConvertTimeToString

Converts a time from centiseconds to a string, representing the time in the desired format.

Syntax:

```
string ConvertTimeToString(integer timeInCentiSeconds, StrTimeFormat desiredFormat)
```

See "Time Formats" below for a list of the constants that apply for the *desired-Format* parameter.

ActiveX example:

```
numberOfFrames = SS.ConvertTimeToString(60*100, KTF_NTSC_Frames)
```

Converts one minute (ie, 6000 hundredths of seconds) to the corresponding number of NTSC frames.

Xtra example:

```
put SSConvertTimeToString(10*60*100, #KTF_SMPTE_DropFrame)
```

Displays 10 minutes as an SMPTE (ie, NTSC) dropframe timecode readout.

◆ **NOTE:** Constant names, such as KTF_SMPTE_DropFrame in the example above, must be preceded by a #-sign in Lingo.

Time Formats

The ConvertTimeToString and ConvertStringToTime functions convert between the SMARTSCRIPT time representation, which is centiseconds, and some other commonly used timecode and framecode formats. While the standard time representation makes it convenient to do calculations on time values, it is not very practical from a user's viewpoint. Hence the need to be able to convert between the internal, numeric time representation and other commonly used formats.

The standard time representation is a long integer that specifies the number of hundredths of seconds since midnight; eg, 500 equals five seconds (5 * 100) and 36000 equals 6 minutes (6 * 60 * 100). This is used for example as a parameter to the CueLocateTime command.

All other time and framecode formats are represented as strings. This makes it easy to display them in dialog boxes, or accept them as user input. The following constants are used to specify the string format.

StrTimeFormat Constant	*Description*
KTF_Normal	Normal time display, with hundredths of seconds, as "HH:MM:SS.hh"
KTF_FILM_24	Film time display (24 frames per second).
KTF_EBU_25	EBU time display (25 fps).
KTF_SMPTE_NonDrop	SMPTE non-drop time display (29.97 fps).
KTF_SMPTE_DropFrame	SMPTE dropframe time display (29.97 fps).
KTF_SMPTE_30	SMPTE "black & white" time display (30 fps).
KTF_PAL_Frames	PAL frame-count.
KTF_NTSC_Frames	NTSC 29.97 fps frame-count.
KTF_NTSC_FilmFrames	NTSC 24 film conversion format frame-count (laserdisc).

◆ **NOTE:** For the frame-count formats, the first frame is numbered "1".

See the "TCCalc" form in the Visual Basic sample application for examples on how the conversion functions can be used.

ConvertStringToTime

This function performs the opposite of the ConvertTimeToString, and converts back from a string representation (assumed to be in the specified format) to the corresponding number of centiseconds.

Syntax:

```
integer ConvertStringToTime(string timeInSpecifiedFormat, StrTimeFormat timeFormat)
```

See “Time Formats” on page 65 for a list of the constants that apply for the *timeFormat* parameter.

ActiveX example:

```
SS.CueLocateTime "LDP", SS.ConvertStringToTime(“10:00”, KTF_EBU_25), 0
```

Locates the PAL video frame corresponding to the time 10 minutes.

Xtra example:

```
whereToGo = SSConvertStringToTime(field “TimeEntry”, #KTF_SMPTE_DropFrame)
```

Picks up the user entry from the field named “TimeEntry”, translates it from dropframe format to centiseconds, and stores the result in the variable whereToGo. This value can subsequently be used in a CueLocateTime command to go to that time position.

IgnoreErrors

Controls whether errors in Cue commands are reported at run time. If set to True, the user won't be notified of such errors. If set to False, an error message will be displayed when an error occurs.

Syntax:

```
boolean IgnoreErrors
```

ActiveX example:

```
SS.IgnoreErrors = True
```

Disables error messages originating from Cue commands. In the ActiveX implementation IgnoreErrors is provided as a property, so you can set it either in the Property window or through programming.

◆ **IMPORTANT:** In the ActiveX implementation, setting IgnoreErrors to True disables *all* reporting of cue errors. Thus, you can't use Visual Basic's Err.Number to retrieve the error code. If you want to handle errors yourself, use the On Error statement in Visual Basic to specify your own error handler, and set IgnoreErrors to False.

Xtra example:

```
SSIgnoreErrors TRUE
```

Disables error messages originating from Cue commands. In the Xtra implementation, IgnoreErrors is provided as a command, which means that you can't read its current setting.

LastError

Returns the error code of the most recent SMARTSCRIPT command (see “Error Codes” on page 70), or zero if the last command didn’t cause any error.

◆ **NOTE:** This function is only available in the Xtra implementation. The ActiveX implementation uses the standard mechanisms for returning errors to the host application.

Syntax: integer LastError()

Xtra example: if SSLastError() < 0 then Alert “Invalid entry. Please try again.”

ActiveX example: As mentioned above, the LastError function isn’t available in the ActiveX implementation, as ActiveX contains its own error handling mechanism. Use the appropriate functions in your programming language to access the error code. For example, in Visual Basic you obtain the most recent error from the Number property of the Err object (ie, by referring to Err.Number). In order to use this method, you must also tell Visual Basic that you want to handle errors yourself using an On Error statement.

Error Codes

When commands fail, SMARTSCRIPT tells you what's wrong by displaying an error message containing an error code. If possible, the offending line in your program will be highlighted as well.

The table below lists the error categories, codes and values.

Category	*Code*	*Value*	*Meaning*
Open	-2147220503	1001	System description file not found.
	-2147220502	1002	Bad data in system description file.
	-2147220501	1003	Insufficient memory.
	-2147220500	1004	Too many devices in system description file.
	-2147220499	1005	Too many items in system description file.
	-2147220498	1006	SMARTSCRIPT was already open.
	-2147220403	1101	Specified communications port not available.
Cue	-2147220303	1201	No such device.
	-2147220302	1202	Cue can't be used with specified device.
	-2147220301	1203	Position parameter out of range.
	-2147220300	1204	Level parameter out of range.
	-2147220299	1205	Rate parameter out of range.
	-2147220298	1206	No such mode or state.
	-2147220297	1207	Value out of range for specified state.
Convert	-2147220203	1301	Invalid time.

The category indicates the circumstances under which errors can occur; errors in the Open category can occur when opening SMARTSCRIPT (ie, using the Open command), errors in the Cue category comes from the various Cue commands, etc.

When an error occurs, an error message will be displayed and the offending line will be selected (if possible). The error message shows the error code and in some cases some additional information.

◆ **NOTE:** Errors in the Cue category can be disabled using the IgnoreErrors property/command.

Error Code and Value

The table above shows both an error code and an error value. As you can see, the error value closely follows the category and error within each category. However, in order to adhere to the standard for encoding errors that occur in ActiveX and Xtra modules, error codes must be constructed in a specific way. Error codes are 32 bit values where the upper 16 bits must contain 0x8004 (hex). Thus, the error code returned by SMARTSCRIPT is composed by assembling the error value in the lower 16 bits with the value 0x8004 in the upper 16 bits. The result of this operation is the error code, as shown in the table. The error code will come out negative if the resulting 32 bit number is shown as a signed decimal number. In some cases, the error code may also be displayed in hex.

Handling Errors

While the above table may be helpful during debugging of your SMARTSCRIPT applications, such error codes won't make any sense to the user of your finished application. Although your application, once debugged, may not generate any errors of its own, errors may result from changes to the system configuration, or as the result of user input.

For example, if the system description file is moved or renamed, error code -2147220503 (System description file not found) will be returned from the Open command. Likewise, if you use the ConvertStringToTime function to convert a value entered by the user in a dialog box, it will return error code -2147220203 (Invalid time) if the string didn't contain a valid time string.

In cases where you anticipate such errors, you can design your application to handle them gracefully. In Visual Basic, you can use the On Error statement to specify your own error handler. You can then retrieve the error code from Err.Number. You can use the And operator to convert the error code to the corresponding error value:

```
ErrorValue = Err.Number And &HFFFFF
```

▼ **IMPORTANT:** Setting the IgnoreErrors property to True in the ActiveX implementation masks all Cue errors so they won't be reported back to your application. If you want to handle such errors yourself, make sure that the IgnoreErrors property is set to False.

Index