

Using Scripts

The DVD-Video specification includes a simple yet powerful scripting language. DVD Studio Pro gives you full access to this capability. You can add sophisticated interactivity and control to a project with only a few simple scripts.

Scripts in DVD Studio Pro are created as separate items within a project, so they can be assigned easily to any item that supports a script. Scripts can be assigned to buttons or attached to the start or end of any track, story, menu, or marker.

With scripts you can

- allow viewers choices that affect what buttons, menus, and tracks they see
- add random play to your project
- allow viewers to create a playlist of favorite tracks

Creating a Script

You use the script editor to enter the text of your scripts.

To open the script editor:

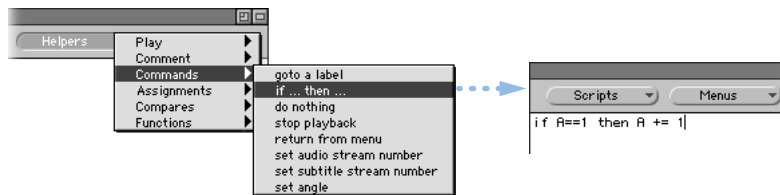
- 1 Double-click a script tile.



The script editor appears.



- 2 To enter a script, do one of the following:
 - Type in the editor window.
 - Choose from menus to automatically enter commands and names of items in your project.



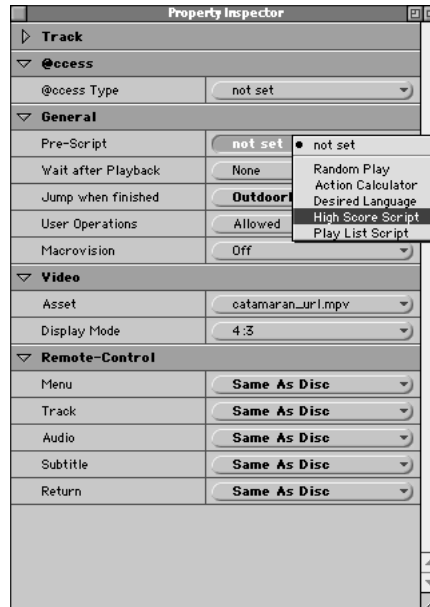
- 3 Click OK or close the script editor to format the script.

Assigning Scripts

Once you create a script, you can assign it as the action for a button, track, menu, marker, story, slideshow, or remote control key.

To assign a script:

- 1 Create the script.
- 2 Select the item to which you want to assign the script.
- 3 In the Property Inspector, choose the script from the appropriate pop-up menu.



You can assign scripts to control the following types of actions:

- A pre-script runs before its assigned item is displayed.
- A timeout action runs if the viewer does not choose an option within a preset period of time.
- A script assigned to a remote control key runs if the viewer clicks that key.
- An action runs when an item (such as a button or menu) is displayed or activated.

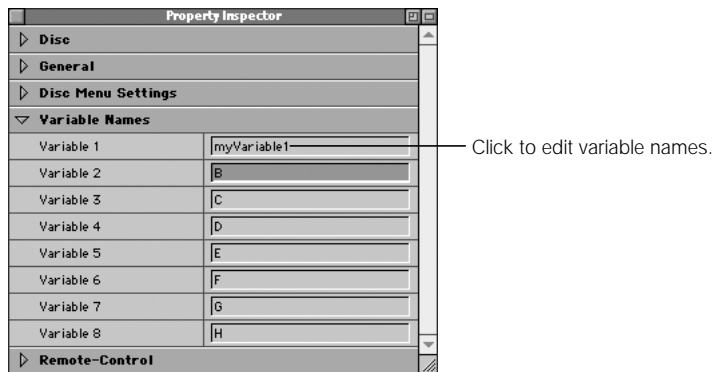
Assigning Global Variables for Scripts

You can create and name up to eight global variables (also called GPRMs) for scripts. The variables store data used by scripts. To create a variable, you use it in a script or name it in the Property Inspector. The variables are stored as properties of the disc.

Important DVD players reset the content of all variables whenever the viewer uses the Time Search or Time Play functions of the remote control. If your project depends on maintaining variable values, you should prohibit use of these functions. To do so, view the disc in the Property Inspector, open the Remote-Control area, and choose Prohibited from the Time Search / Time Play pop-up menu.

To view global script variables:

- Select the disc to see its properties in the Property Inspector.



Viewing Script Properties

To view or change information about a script:

- 1 Select a script.
- 2 In the Property Inspector, make the changes you want.



The DVD-Video standard limits the number of commands in a script to 128. One line of a script is roughly equivalent to one command. The Commands Used and Commands Free values give an indication of the size of the script.

Scripting Reference

About Registers, Parameters, and Variables

Variables are called *registers* or *parameters* in the parlance of DVD-Video players. Each player has several registers or parameters. Each register can hold a positive, whole-number value between 0 and 65535.

There are two kinds of registers:

- *System parameters*: Hold information about languages, tracks, and subtitle and audio streams. You can use system parameter information through routines built into the scripting language.
- *General-purpose parameters*: Can be used by an application created with DVD Studio Pro for such features as auto-selection of buttons. There are eight general-purpose parameters.

Registers work like global variables. You can use them in any script. Any change to a register is reflected at once in all scripts.

When a disc starts, the registers are set to zero. The values of the registers are lost if the viewer ejects or stops the disc or turns off the player. In this manual, the values of registers or parameters are called *variables*.

Limitations of Variables and Registers

- DVD-Video players don't support recursion or local variables.
- Registers have no overflow or underflow flags.
- If the script tries to store a value smaller than 0 or larger than 65535, no error message is given; instead, the value "wraps around." For example, if you subtract 7 from 3, you would normally get -4. But if you do this in a script, 65532 will be stored. If you add 6 to 65534 in a script, 4 is stored instead of 65540.

Depending on the application design, plausibility checks may be necessary. You cannot create compound statements. For example,:

The first line must be written as:	<pre>B = getAudioStream() if A == B then nop</pre>
------------------------------------	--

The second line must be stated as:	<pre>A = B A += C</pre>
------------------------------------	-----------------------------

This code is not allowed:	<pre>if A == getAudioStream() then nop A = B + C</pre>
---------------------------	--

Script Reference Conventions

In this scripting reference, the eight registers available for scripting are called *variables*.

"Variable" stands for one of your variables.

"Constant" stands for an integer value between 0 and 65535.

"Function" stands for any of the built-in functions.

Items enclosed in square brackets are options. For example, the following code means that you can either use a variable or a constant after the = sign:

```
variable = [variable|constant]
```

Numbers

There are three ways to enter numbers in scripts:

Decimal numbers are written directly in the script.	<pre>A = 10</pre>
---	-------------------

Hexadecimal numbers must begin with a dollar (\$) sign.	<pre>A = \$FFE3</pre>
---	-----------------------

Binary numbers have a leading percentage (%) sign.	<pre>A = %00011101</pre>
--	--------------------------

Labels

Any line in a script can have a label associated with it. You can jump to a label with the `gotoLabel` command. A label must begin with a letter.

Example	<code>label: A= 5</code>
---------	--------------------------

Comments

A line with a `#` sign at the beginning is not executed. You can use these lines to write notes about the script or to disable certain lines of the script.

Example	<code># This is a comment.</code>
---------	-----------------------------------

Functions and Commands

Operators

Assignment

Assigns the value on the right to the variable on the left. The value can come from another variable, a constant, or a built-in function.

Syntax	<code>variable = [variable constant function]</code>
--------	--

Examples	<code>A = B</code>
	<code>A = 5</code>
	<code>A = getAudioStream()</code>

Operators With Assignment

Addition

Adds the value from the right to the variable on the left.

Syntax	<code>variable += [variable constant]</code>
--------	--

Examples	<code>A += B</code>
	<code>A += 7</code>

Subtraction

Subtracts the value on the right from the variable on the left.

Syntax	variable -= [variable constant]
--------	-----------------------------------

Examples	A -= B
	A -= 3

Multiplication

Multiplies the variable on the left by the value on the right; stores the result in the variable on the left.

Syntax	variable *= [variable constant]
--------	-----------------------------------

Examples	A *= B
	A *= 8

Division

Divides the variable on the left by the value on the right; stores the result in the variable on the left. The result is truncated to an integer.

A division by zero results in \$FFFF or 65535.

Syntax	variable /= [variable constant]
--------	-----------------------------------

Examples	A /= B
	A /= 19

Modulo

Divides the variable on the left by the value on the right; stores what's left over (modulo) in the variable on the left.

Syntax	variable %= [variable constant]
--------	-----------------------------------

Examples	A %= B
	A %= 20

Bit-wise And

Does a “bit-wise and” operation with the two operands.

Syntax	<code>variable &= [variable constant]</code>
--------	--

Examples	<code>A &= B</code> <code>A &= 6532</code>
----------	---

Bit-wise Or

Does a “bit-wise or” operation with the two operands.

Syntax	<code>variable = [variable constant]</code>
--------	--

Examples	<code>A = B</code> <code>A = 456</code>
----------	--

Bit-wise Exclusive Or

Does a “bit-wise exclusive or” operation with the two operands.

Syntax	<code>variable ^= [variable constant]</code>
--------	--

Examples	<code>A ^= B</code> <code>A ^= 21</code>
----------	---

Random Number

Generates a random number between 1 and the value on the right.

Syntax	<code>variable ?= [variable constant]</code>
--------	--

Examples	<code>A ?= B</code> <code>A ?= 34</code>
----------	---

Functions

getAudioStream()

Yields the current audio stream number (between 1 and 8).

Syntax	<code>variable = getAudioStream()</code>
--------	--

Example	<code>A = getAudioStream()</code>
---------	-----------------------------------

getSubtitleStream()

Yields the current subtitle stream number (between 1 and 32).

Syntax	<code>variable = getSubtitleStream()</code>
--------	---

Example	<code>A = getSubtitleStream()</code>
---------	--------------------------------------

getRegionCode()

Yields the region code of the player. This is a binary value. Bit 0 means region 1, bit 1 means region 2, and so on.

Syntax	<code>variable = getRegionCode()</code>
--------	---

Example	<code>A = getRegionCode()</code>
---------	----------------------------------

getCurrentItem()

Yields the current item (usually the item the script is attached to: a script, if the script was called directly, or a menu or track, if the script was assigned as a pre-script).

Syntax	<code>variable = getCurrentItem()</code>
--------	--

Example	<code>A = getCurrentItem()</code>
---------	-----------------------------------

getLastItem()

Yields the last item played.

Syntax	<code>variable = getLastItem()</code>
--------	---------------------------------------

Examples	<pre>A = getLastItem() if A == Track "drehung shell.dvd.mpv Track" then nop</pre>
----------	---

getCurrentTrack()

Yields the track currently playing or the track that was playing before a jump to a menu.

Syntax	<code>variable = getCurrentTrack()</code>
--------	---

Example	<code>A = getCurrentTrack()</code>
---------	------------------------------------

Procedures

setAudioStream

Sets the number of the audio stream to be played.

Syntax	setAudioStream [constant]
--------	---------------------------

Example	setAudioStream 5
---------	------------------

setSubtitleStream

Sets the number of the subtitle stream to be displayed.

Syntax	setSubtitleStream [constant]
--------	------------------------------

Example	setSubtitleStream 19
---------	----------------------

Commands

nop

This command does nothing. It serves as a placeholder.

Syntax	nop
--------	-----

Example	if A == 5 then nop
---------	--------------------

stop

This command is equivalent to clicking the Stop key on the remote control.

Syntax	stop
--------	------

Example	stop
---------	------

exitScript

This command, which only works with pre-scripts, exits from the running script immediately. The player goes to the next item to which the pre-script is assigned.

Syntax	exitScript
--------	------------

Example	exitScript
---------	------------

gotoLabel

This command jumps to another position in the script.

Syntax	<code>gotoLabel [label]</code>
Example	<code>gotoLabel here_next</code>

return

This command returns from a menu to the player's position before the jump to the menu. (The player continues playing the track as if nothing happened.)

Syntax	<code>return</code>
Example	<code>return</code>

play

This command plays an object. Objects are tracks, markers, menus, buttons, and scripts.

<code>play Track</code>	Plays the track from the start.
<code>play Marker "myMarker" of Track "myTrack"</code>	Plays the track starting from the marker.
<code>play Menu</code>	Evaluates all the selection conditions.
<code>play Button</code>	When this menu appears, this button is selected. Overrides selection conditions normally assigned to the button.
<code>play Script</code>	Calls that script, stopping the execution of the current script.

Syntax	<code>play [object]</code>
Examples	<pre>play Track "myTrack" play Marker "myMarker" of Track "myTrack" play Menu "MyMenu" play Button "myButton" of Menu "myMenu" play Script "myScript" A = getLastItem() if A == Menu "shell step 01 Menu" then play Button "1step" of Menu "5T84"</pre>

Control Structures

if then

Syntax	if variable [operator] [variable constant] then [assignment procedure command]
--------	---

The “if” clause enables comparisons and the execution of commands based on the result of comparisons.

Several operators are available for the if clause:

Equal

This operator compares the values on both sides of the operator and executes the “then” condition if they are equal.

Syntax	variable == [variable constant]
--------	-----------------------------------

Example	if A == B then nop
---------	--------------------

Not Equal

This operator compares the values on both sides of the operator and executes the “then” condition if they are not equal.

Syntax	variable != [variable constant]
--------	-----------------------------------

Example	if A != B then play Track “myTrack”
---------	-------------------------------------

Using NVRAM in DVD Studio Pro

DVD Studio Pro supports the Philips Professional DVD Player 170. This player has NVRAM, or non-volatile memory, which means variables remain stored when the player is turned off. To use the NVRAM feature with DVD Studio Pro, the player must have firmware version 9.25 or later installed. Upgrade information is available from Philips.

The NVRAM has 256 addresses from 0 to 255. Every address has a size of 16 bits.

When Use NVRAM is turned on in the Preferences and you use NVRAM commands in your project, DVD Studio Pro creates a folder called PROF containing a file named INFO.ID. This file contains the NVRAM information for the disc. Do not edit this file. When you multiplex your disc, make sure the folder is stored in the same folder as the VIDEO_TS directory.



When you choose Build & Format Disc from the File menu, DVD Studio Pro automatically includes the PROF folder in the proper place on the disc.

When you preview your project, the debugging window shows all current values stored in the NVRAM. (See "Debugging in Preview Mode" on page 116.)

To store information in or read information from the NVRAM, you use four simple commands in your scripts. (Using NVRAM commands on a player without installed NVRAM has no effect.)

poke

This command stores the value of the first global variable (by default, this variable is named `A`; you can change the name in the disc properties) in the NVRAM at the address specified.

Syntax	<code>poke [address]</code>
--------	-----------------------------

Example	<code>poke 33</code>
---------	----------------------

pokeAll

This command stores the values of all global variables (by default, these variables are named *A* through *H*; you can change the name in the disc properties) in the NVRAM. *A* is stored in the address specified, *B* is stored in the next address, and so on.

Syntax	<code>pokeAll [address]</code>
--------	--------------------------------

Example	<code>pokeAll 33</code>
---------	-------------------------

peek

This command retrieves the value from the NVRAM address specified and writes it to the corresponding global variable. For example, if you store variables *A–H* in addresses 33–40 and you use the command `peek 34`, the command retrieves the value of address 34 and writes it to variable *B*.

Syntax	<code>peek [address]</code>
--------	-----------------------------

Example	<code>peek 33</code>
---------	----------------------

peekAll

This command retrieves the contents of eight sequential addresses beginning with the address specified and writes the contents in the same order in the global variables *A–H*.

Syntax	<code>peekAll [address]</code>
--------	--------------------------------

Example	<code>peekAll 33</code>
---------	-------------------------

Important To avoid loss of information, after using the `poke` or `pokeAll` command, instruct the player to wait at least one second before using it again. The player needs this time to store the information safely.

Determining Whether NVRAM Is Available

You can use this simple test procedure to find out whether NVRAM is active. Script 1 writes the value 5 into the NVRAM at address 33. Script 2 sets variable A to 0 and then reads the value in address 33 and stores it in variable A. If the value remains at 0, then NVRAM is not available.

- 1 Create two simple menu graphics and name them *NVRAM installed* and *NVRAM not installed*.
- 2 Create two scripts:

Script 1:

```
#storing A in the NVRAM
A = 5
Poke 33
Script 2:
```

```
#retrieving value of location 33 and storing it in variable A:
```

```
A = 0
Peek 33
if A==0 then play menu "NVRAM not installed"
play menu "NVRAM installed"
```

- 3 Assign the scripts to a menu.
- 4 Preview the menu.