

New Technical Notes

Macintosh



Developer Support

Script Manager Variables

Text

M.TE.ScriptVars

Written by: John Harvey & Peter Edberg

June 1989

This Technical Note describes, in detail, the local and global script variables.

Introduction

The Script Manager maintains a number of global variables which can be read with the routine `_GetEnviron`s. These variables can be set by a corresponding routine, `_SetEnviron`s. In addition, each script interface system maintains variables of its own. These are referred to as local variables in *Inside Macintosh*, Volume V-293, The Script Manager, and are read by `_GetScript` and set by `_SetScript`.

Think of it like this: the Script Manager maintains an environment in which different script interfaces can run. The global variables are used to set up and maintain the environment (thus the names for the routines `_GetEnviron`s and `_SetEnviron`s), and the local variables control how the script itself works (so we have `_GetScript` and `_SetScript`).

Global Variables

When you call `_GetEnviron`s or `_GetScript`, you describe the variable you are interested in with a verb. A verb is simply an integer constant which the Script Manager uses to figure out which variable you want to read or set. The table in *Inside Macintosh*, V-313, gives incorrect names and descriptions for some of the `_GetEnviron`s and `_SetEnviron`s verbs. Table 1 provides correct descriptions.

Constant	Value	Meaning
smVersion	0	Script Manager version number
smMunged	2	Global modification count
smEnabled	4	Script count; 0 if Script Manager not enabled
smBidirect	6	Bidirectional script flag
smFontForce	8	Force font flag
smIntlForce	10	Force international utilities flag
smForced	12	Current script forced to system script
smDefault	14	Current script defaulted to Roman script
smPrint	16	Print action vector
smSysScript	18	Preferred system script
smLastScript	20	Last keyboard script
smKeyScript	22	Keyboard script
smSysRef	24	System folder volRefNum
smKeyCache	26	[Obsolete, do not use]
smKeySwap	28	Keyboard swapping resource handle
smGenFlags	30	General flags
smOverride	32	Script override flags
smCharPortion	34	Ch vs Sp Extra proportion, 4.12 fixed

Table 1—Verbs for `_GetEnviron`s and `_SetEnviron`s

The descriptions in the table are still a bit sketchy. The next section describes each variable in more detail and describes the size of each global.

Byte or word globals are mapped to the low-order byte or word of the `LongInt` returned by `_GetEnviron`s, with the high-order parts set to zero. Similarly, for these globals `_SetEnviron`s ignores all but the appropriate part (low-order byte or word) of its params value.

Verb Name	Bytes	Brief Description
smVersion	2	Script Manager version number

At boot time, the version global is initialized to the value `SMgrVers`. The high byte is the major version number and is defined in the MPW interface files. The low byte is updated when any changes are made to the Script Manager.

At boot time or switch-launch time, the enabled global is initialized to zero, then incremented for each script that is installed and enabled. Since the Roman script system should always be installed by the Script Manager, a value of zero indicates that the Script Manager is not enabled.

It should be noted that older versions of the Script Manager treated this as a Boolean. In other words, if there was more than one script installed, `_GetEnvirons(smEnabled)` would return 255 (when `_GetEnvirons` returns a Boolean value \$FF represents true).

For this reason, when testing to see if more than one script is installed, it is best to test as follows:

```
scriptsininstalled := GetEnvirons(smEnabled);
IF scriptsininstalled > 1 THEN
{more than one script available, use Chartype, etc.}
```

<code>smBidirect</code>	1	Bidirectional script flag
-------------------------	---	---------------------------

The bidirectional global indicates that at least one bidirectional script is installed. It should be set to true (\$FF) by the Arabic and Hebrew script systems. This is not presently done, but will be corrected in future versions of these systems.

<code>smFontForce</code>	1	Force font flag
<code>smIntlForce</code>	1	Force international utilities flag
<code>smForced</code>	1	Current script forced to system script
<code>smDefault</code>	1	Current script defaulted to Roman script

At boot time, `FontForce` and `IntlForce` are set from the 'itlc' resource, and `Forced` and `Default` are set to zero. These are all flags with the value zero for false and \$FF for true. `FontForce` and `IntlForce` control the operation of the `_FontScript`, `_Font2Script`, and `_IntlScript` routines. `Forced` and `Default` report the actions of these routines.

Setting `FontForce` to true forces Roman fonts to be interpreted as belonging to the system script. This is for compatibility with applications that hard-code font numbers.

`IntlForce` determines the behavior of the `_IUGetIntl` call. When `intlforce` is set to true, `_IUGetIntl` will return a handle to the international resources (of type 'itlx' where x is 0-2) for the system script. When `IntlForce` is false, the `_IUGetIntl` will use the font of the current port to determine the appropriate resources to fetch. Thus date formats, sorting, etc. can reflect the current script.

<code>smPrint</code>	4	Print action vector
----------------------	---	---------------------

Print action routine vector; set up at boot time. See `M.TE.PrintAction`.

<code>smSysScript</code>	2	Preferred system script
<code>smLastScript</code>	2	Last keyboard script
<code>smKeyScript</code>	2	Keyboard script

At boot time and switch-launch time, `SysScript` and `KeyScript` are set from the `SysScript` field of the 'itlc' resource if that script is installed and enabled; otherwise, `SysScript` and `KeyScript` are set to Roman (without setting `Default`).

smScriptJust	8	Justification Flag
smScriptRedraw	10	Word Redraw Flag
smScriptSysFond	12	Preferred System Font
smScriptAppFond	14	Preferred Application Font
smScriptNumber	16	Script 'itl0' ID
smScriptDate	18	Script 'itl1' ID
smScriptSort	20	Script 'itl2' ID
smScriptFlags	22	Script Flags Word (new)
smScriptToken	24	'itl4' ID number (new)
smScriptRsvd	26	Reserved
smScriptLang	28	Script's language code (new)
smScriptNumDate	30	Number/date representation codes (new)
smScriptKeys	32	Script 'KCHR' ID
smScriptIcon	34	Script 'SICN' ID
smScriptPrint	36	Script printer action routine
smScriptTrap	38	Trap entry pointer
smScriptCreator	40	Script file creator
smScriptFile	42	Script file name
smScriptName	44	Script name

Table 2—Local Variable Constants

Here again the descriptions are a little terse. The following section describes each variable in more detail and describes the size of each variable.

Verb Name	Bytes	Brief Description
smScriptVersion	4	Script Interface version number

When the script interface is loaded, this is set to the current version number.

smScriptMunged	2	Local modification count
----------------	---	--------------------------

This variable is incremented each time `_SetScript` is called.

smScriptEnabled	1	Script Enabled Flag
-----------------	---	---------------------

A Boolean which indicates whether the script has been enabled. Set to \$FF when enabled

and zero when not enabled.

smScriptRight 1 Right to Left Flag

A Boolean indicating if text should be drawn right to left or left to right. It is set to \$FF for right to left text (Arabic and Hebrew scripts) and zero for left to right (Roman).

smScriptJust 1 Justification Flag

A byte flag which describes how text should be justified. The possible settings correspond to the justification flags used by TextEdit.

- 0 = left justification
- 1 = center justified
- 1 = right justified

smScriptRedraw 1 Word Redraw Flag

A byte flag describing how much of a line should be redrawn when text is being entered.

- 0 Only draw a character
- 1 Redraw the entire word
- 1 Redraw the entire line (Arabic)

smScriptSysFond 2 Preferred System Font

This is the font family ID for the preferred System Font. In a Roman system, ScriptSysFond is 0, the family ID for Chicago.

smScriptAppFond 2 Preferred Application Font

Font family ID for the preferred Application Font. In a Roman system, ScriptAppFond is 3, the family ID for Geneva.

smScriptNumber 4 Script 'itl0' ID

Resource ID of 'itl0' for this script. The 'itl0' resource describes how numbers and times should be displayed. The resource ID should match the country version code for a given country.

smScriptDate 4 Script 'itl1' ID

Resource ID of the 'itl1' for this script. The 'itl1' describes how dates should be displayed.

smScriptSort 4 Script 'itl2' ID

Resource ID of the 'itl2' for this script. The 'itl2' contains routines for sorting. See M.TE.NewStringComp.

smScriptFlags 2 Script flags

This verb provides access to the script flags word, which contains bit flags that describe

features of the script. This word is initialized from the script's 'itlb' resource. Constants specifying the bit numbers are described in Table 3.

Who Does What?

Table 4 breaks the documented routines into common Script Manager routines and interface specific routines.

Common Routines	Interface Supplied
_FontScript	_CharByte
_IntlScript	_CharType
_KeyScript	_Pixel2Char
_GetEnvirons	_Char2Pixel
_SetEnvirons	_Transliterate
_Font2Script	_FindWord
_Format2Str	_HiliteText
_FormatStr2X	_DrawJust
_FormaX2Str	_MeasureJust
_GetFormatOrder	_ParseTable
_InitDateCache	_VisibleLength
_IntlTokenize	_FindScriptRun
_LongDate2Secs	_PortionText
_LongSecs2Date	
_Str2Format	
_String2Date	
_String2Time	
_StyledLineBreak	
_ToggleDate	
_ValidDate	

Table 4—Script Manager Routines and Interface Specific Routines

`_GetScript` and `_SetScript`, which return the values of local script variables, are implemented by the Script Manager for some verbs and the script interface system for others.

There is also a group of Script Manager routines which don't use the `_ScriptUtil` trap, but are documented in The Script Manager chapter of *Inside Macintosh*, Volume V-293 or The Script Manager 2.0 Interim Chapter. These routines are utilities that read and write to low-memory or PRAM. It is important to use these routines when they are available. That will allow Apple to modify where global variables, etc. are stored, and your application will remain compatible. The utilities are:

```
GetDefFontSize
GetSysFont
GetAppFont
GetMBarHeight
GetSysJust
SetSysJust
```

ReadLocation (documented in Interim Chapter)

WriteLocation (documented in Interim Chapter)

Further Reference:

- *Inside Macintosh*, Volume V-293, The Script Manager
- M.TB.KeyMapping
- M.TE.PrintAction