NAME

FvwmScript - module to build graphic user interface

SYNOPSIS

FvwmScript must be spawned by Fvwm. It will not work from the command line.

DESCRIPTION

FvwmScript is a module which allows you to build many graphical applications such as desktop accessories, button panel with pop up menus, modal dialogs... At the startup, FvwmScript reads the file which is specified on the command line. This file contains the script. This script is not included in the configuration file of Fvwm.

An FvwmScript script is fully controllable by using the keyboard. (Shift)-Tab circulates around the widgets, Return simulates a mouse click, the arrows move the cursor or change the values of the widget and Escape "cancels" for Menu and PopupMenu.

INVOCATION

FvwmScript can be invoked by inserting the line ‘Module FvwmScript name_of_script’ in the .fvwm2rc file. The file "name_of_script" can start with a slash, in which case, it’s a fully qualified path, and the file is read. If "name_of_script" does not start with a slash, FvwmScript will look in a few different places. If the .fvwm2rc contained the command line ‘*FvwmScript: Path path_of_the_script_directory’, FvwmScript will try that directory. If that doesn’t work, FvwmScript tries the system configuration directory and the user configuration directory as described under the "Read" command in the fvwm man page.

The command to start FvwmScript can be placed on a line by itself, if FvwmScript is to be spawned during fvwm’s initialization, or can be bound to a menu or mouse button or keystroke to invoke it later.

CONFIGURATION OPTIONS

The following commands can be used in the config file (see *fvwm*(1), section **MODULE COMMANDS** for details). They are used only if the corresponding script commands are not used in the script.

*FvwmScript: DefaultFont *font*

Specifies the default font to be used. If not specified with this command or in the script with the Font command, fixed font is assumed.

*FvwmScript: DefaultFore *color*

Specifies the default foreground color to be used. If not specified with this command or in the script with the ForeColor command, black is used.

*FvwmScript: DefaultBack *color*

Specifies the default background color to be used. If not specified with this command or in the script with the BackColor command, grey85 is used.

*FvwmScript: DefaultHilight *color*

Specifies the default hilight color to be used. If not specified with this command or in the script with the HilightColor command, grey100 is used.

*FvwmScript: DefaultShadow *color*

Specifies the default shadow color to be used. If not specified with this command or in the script with the ShadowColor command, grey55 is used.

*FvwmScript: DefaultColorset *colorset*
: Tells the module to use colorset *colorset* as the default colorset.

ANATOMY OF A SCRIPT

FvwmScript uses a particular programming language. A script is composed of five parts. Heading contains general characteristics of the window and default properties for all widgets. The second part contains instructions whom are executed at the startup of the script. The third part contains periodic tasks which are executed every second. The fourth part contains instructions which are executed at exit. And the last part contains the description of widgets. A widget consists of eleven types of items: text labels, single-line text inputs, radio buttons, checkbox, push buttons, horizontal and vertical scrollbars, rectangles, pop up menus, swallowexecs and mini scrollbars.

HEADING OF A SCRIPT

The syntax is as follows:

WindowTitle *string*
: This option sets the window title.

WindowSize *width height*
: This option sets window size. *width* and *height* are numerical value.

WindowPosition *x y*
: This option sets window position. *x* and *y* are numerical value.

ForeColor {*color*}
: This option sets the default foreground color for all widgets.

BackColor {*color*}
: This option sets the default background color for all widgets.

HilightColor {*color*}
: This option sets the default hilight color for all widgets.

ShadowColor {*color*}
: This option sets the default shadow color for all widgets.

Colorset {*n*}
: This option sets the default colorset for all widgets.

Font {*font*}
: This option sets the default font for all widgets.

UseGettext [*locale_path*]
: Enable the use of the gettext mechanism which is used by the WindowLocaleTitle, LocaleTitle, ChangeLocaleTitle instructions and the Gettext function. If no argument is given, the default FvwmScript locale catalog is used. This catalog is under the locale fvwm installation directory and the text domain is FvwmScript (install_prefix/share/locale/*/LC_MESSAGES/FvwmScript.mo). You can reset this catalog or add some catalogs exactly in the same way than with the **LocalePath** fvwm command (see the fvwm manual page). This instruction should be placed before the WindowLocaleTitle instruction.

WindowLocaleTitle *string*
This option sets the window title, but use the locale catalog(s) defined with UseGettext.

INITIALISATION

This part contains instructions which will be executed at the startup. For example:

```
Init
 Begin
  Do "Exec cat tada.voc > /dev/dsp"
  WarpPointer 1
  Set $ToDo=Restart
 End
```

These instructions are used to play a sound, move the pointer to widget 1 and to initialize $ToDo to "Restart" at every startup.

PERIODIC TASKS

This part of the script contains instructions that are executed every second. For example:

```
PeriodicTasks
 Begin
  If (RemainderOfDiv (GetTime) 10)==0 Then
   Do {Exec xcalc}
 End
```

This example shows how to launch xcalc every 10 seconds.

THE QUIT FUNCTION

This part of the script contains instructions that are executed when the script exits (after the Quit instruction or if you close the window with the Close, Delete or Destroy fvwm command). For Example

```
QuitFunc
 Begin
  Do {Echo bye, bye}
 End
```

Be aware that if you used the KillModule fvwm command to close the script, some instructions or functions which rely on the existence of a communication link between the script and fvwm will not be executed (for example the Do command). To smoothly kill a script with an fvwm command see the **COMMANDS** section.

MAIN OF A SCRIPT

The second part of the script contains the description for every widget in the script. Each widget description has two parts. The first part describes initial properties, the second part contains instructions that are executed when the widget receives messages. All widgets can send and receive messages. All messages are identified by a number. The message "UserAction" is sent to a widget when the user operates the widget. The syntax for the first part is:

```
Widget            id    # A number between 1 and 999 inclusive
Property
 Type       string
 Size width height
 Position   x y
 Title            { string }
 Value            int
 MaxValue   int
 MinValue   int
 Font       string
 ForeColor  { color }
 BackColor  { color }
 HilightColor     { color }
 ShadowColor      { color }
 Colorset   int
 Flags            flagsOpt
```

The flagsOpt option to Flags is a space separated list containing one or more of the keywords *Hidden*, *NoReliefString*, *NoFocus*, *Left* / *Center* / *Right*. *Hidden* is used to specify if the widget is hidden at startup. *NoReliefString* specifies if strings are drawn with relief or not. *NoFocus* specifies if the widget can get the keyboard focus or not. By default all widgets take focus, except Rectangle, HDipstick and VDipstick which cannot. Moreover, the NoFocus widgets are skipped when you circulate around the widgets with the (Shift-)Tab short cut. *Left* / *Center* / *Right* specifies the te xt position. These apply only to ItemDraw, List, Menu, PopupMenu and PushButton. The default is*Center* for ItemDra w and PushButton and *Left* for the other widgets.

LocaleTitle can be used in place of Title, for using the locale catalog(s) defined with UseGettext.

The position of every widget must be specified.

The syntax for the second part is:

```
Main
 Case message of
  SingleClic:
  Begin
   # list of instructions which will be
   # executed when widget receives
   # message "SingleClic". This message is
   # generated by the user.
  End
  1 :
  Begin
   # list of instructions which will be
   # executed when widget receives
   # message 1
  End
 End
```

LIST OF WIDGETS

There is fifteen types of widgets.

CheckBox: Display check box with a string.

Title: title of the check box.

Value: if Value is equal to 1, the box is checked else it is not.

The **Size** property is ignored.

HDipstick: Display a horizontal dipstick.
This widget can be used to display disk usage.

Value: specify the current value of the dipstick.

MinValue: specify the minimum value of the dipstick.

MaxValue: specify the maximum value of the dipstick.

A minimum size of 30x11 is imposed.

HScrollBar: Display an horizontal scrollbar.

Value: position of the thumb.

MaxValue: upper limit of Value.

MinValue: lower limit of Value.

The height property is ignored and a minimum width is imposed. The width should be at least the range plus 37 if all values are to be selectable e.g. a min of 0 and max of 10 has a range of 11 and therefore should have a minimum width of 48.

ItemDraw: Display an icon and/or a string.

Title: string to display.

Icon: icon to display.

MaxValue: x coordinate of the cursor.

MinValue: y coordinate of the cursor.

The size is made large enough to contain the title and/or the icon.

List: Display a list.
List lets user to choose between various options.

Value: specify which option is selected.

MinValue: First visible option.

Title: title contains options displayed in the list. The syntax is the following: {Option 1|Option 2|...|Option N}. All menus are displayed at the top of window.

A minimum height of three items is imposed and the width is made to be at least 108.

Menu: Display a menu whom lets user to choose a option.
Items of type Menu are layed out from left to right along the top of the window. The size and position properties are ignored.

Value: specify which option is selected.

Title: title contains options displayed in the menu. The syntax is the following: {Option 1|Option 2|...|Option N}.

MiniScroll: Display a very small vertical scrollbar.

Value: position of the thumb.

MaxValue: upper limit of Value.

MinValue: lower limit of Value.

The size is set to 19x34.

PopupMenu: Display a pop up menu.

Value: specify what option is selected.

Title: the title has the following syntax: {Option 1|Option 2|...|Option N}."Option 1|Option 2|...|Option N" is the pop up menu which is displayed when pressing mouse button.

The size property is ignored.

PushButton: Display push button with an icon and/or a string.

Title: this string has the following syntax {Title of the button|Option 1|Option 2|Option3|...|Option N}. "Option 1|Option 2|...|Option N" is the pop up menu which is displayed when pressing the right button.

Icon: icon to display.

The button is made large enough to fit the icon and or label.

RadioButton: Display radio button with a string.

Title: title of the radio button.

Value: if Value is equal to 1, the box is checked else it is not.

The size property is ignored

Rectangle: Display a rectangle.
This type of widget can be used to decorate window.

SwallowExec
This type of widget causes FvwmScript to spawn an process, and capture the first window whose name or resource is equal to Title, and display it in the script window.

Title: specify the window name which be captured and displayed in the script window.

SwallowExec: specify the command line to execute to spawn the process. Modules can also be swallowed.

Value: specify the looking of the border. Possible value: -1, 0, 1.

The size is made to be at least 30x30

TextField: Display a text input field.
The text input field can be used to edit a single-line string.

Title: content of text field.

Value: position of the insert point.

MinValue: position of the end of the selection.

MaxValue: first visible character of the title

The height property is ignored, the width is made to be at least 40 pixels wider than the initial contents.

VDipstick: Display a vertical dipstick.

Value: specify the current value of the dipstick.

MinValue: specify the minimum value of the dipstick.

MaxValue: specify the maximum value of the dipstick.

The size is made to be at least 11x30.

VScrollBar: Display a vertical scrollbar.

Value: position of the thumb.

MaxValue: upper limit of Value.

MinValue: lower limit of Value.

The width property is ignored and a minimum height is imposed. The height should be at least the range plus 37 if all values are to be selectable e.g. a min of 0 and max of 10 has a range of 11 and therefore should have a minimum height of 48.

INSTRUCTIONS

Here is the description of all instructions.

HideWidget *id* : hide the widget numbered *id*.

ShowWidget *id*: show the widget numbered *id*.

ChangeValue *id1 id2*
Set the value of the widget numbered *id1* to *id2*.

ChangeMaxValue *id1 id2*
Set the maximum value of the widget numbered *id1* to *id2*.

ChangeMinValue *id1 id2*
Set the minimum value of the widget numbered *id1* to *id2*.

ChangeTitle *id1 id2*
: Set the title of the widget numbered *id1* to *id2*.

ChangeWindowTitle *string*
: Set the title of the window to *string*.

ChangeWindowTitleFromArg *numarg*
: Set the title of the window to the value of the *numarg*-th script argument.

ChangeLocaleTitle *id1 id2*
: As ChangeTitle but use the locale catalog(s) defined with UseGettext.

ChangeIcon *id1 id2*
: Set the icon of the widget numbered *id1* to *id2*.

ChangeForeColor *id1* {*color*}
: Set the foreground color of the widget numbered *id1* to {*color*}.

ChangeBackColor *id1* {*color*}
: Set the background color of the widget numbered *id1* to {*color*}.

ChangeColorSet *id1 id2*
: Set the colorset of the widget numbered *id1* to *id2*. Specifying widget 0 sets the main window colorset.

ChangePosition *id1 x y*
: Move the widget numbered *id1* to position (*x*,*y*).

ChangeSize *id1 width height*
: Set the size of the widget numbered *id1* to (*width*,*height*).

ChangeFont *id1 newfont*
: Set the font of the widget numbered *id1* to *newfont*.

WarpPointer *id*
: Warp the mouse pointer into the widget numbered *id*.

WriteToFile *filename* {*str1*} {*str2*} etc
: Write to the file *filename* the string which is the concatenation of all arguments *str1*, *str2*, etc.

Do {*command args*}
: Execute the fvwm command inside the Do block. Any fvwm command as described in the fvwm2 man page can be used. Commands are sent from this module to the fvwm main program for processing. The length of the command and arguments can not exceed 988 characters.

Set $*var*={*str*1} {*str2*} etc
: Concatenate all arguments to a string and set the variable $*var* to this string.

Quit: quit the program.

SendSignal *id1 id2*
: Send a message numbered *id2* to widget *id1*.

SendToScript *id_script* {*str1*1} {*str2*} etc
: Send a message to the script identified by id_script. The message is the concatenation of str1, str2...

Key *Keyname Modifier id sig str1 str2*etc
: Binds a keyboard key to the instruction

  SendSignal *id sig*

  and sets the "last string" to the concatenation of str1, str2... (see the LastString function). The *Keyname* and *Modifiers* fields are defined as in the fvwm Key command.

ARGUMENTS

Most of commands use arguments. There are two kinds of arguments: numbers and strings. A numerical argument is a value which is between -32000 and +32000. A string is always surrounded with braces. Variables always begin with the character "$" and can contain both numbers and strings.

FUNCTIONS

All functions use arguments. Functions can return both a string and a number. The syntax is:

```
(function argument1 argument2 etc)
```

Here is the complete list of arguments:

(GetTitle *id*)
: Return the title of the widget numbered *id*.

(GetValue *id*)
: Return the current value of the widget numbered *id*.

(GetMinValue *id*)
: Return the current Min value of the widget numbered *id*.

(GetMaxValue *id*)
: Return the current Max value of the widget numbered *id*.

(GetFore *id*)
: Return the current RGB foreground value of the widget numbered *id* in the hex format RRGGBB.

(GetBack *id*)
: Return the current RGB background value of the widget numbered *id* in the hex format RRGGBB.

(GetHilight *id*)
: Return the current RGB hilight value of the widget numbered *id* in the hex format RRGGBB.

(GetShadow *id*)
: Return the current RGB shadow value of the widget numbered *id* in the hex format RRGGBB.

(GetOutput {*str*} *int1 int2*)
: Executes the command *str*, gets the standard output and returns the word which is in the line *int1* and in the position *int2*. If *int2* is equal to -1, GetOutput returns the complete line.

(NumToHex *int*)
: Return the hexadecimal value of *int*.

(HexToNum {*str*})
: Return the decimal value of *str*, *str* must be an hexadecimal value.

(Add *int1 int2*)
: Return the result of (*int1*+*int2*).

(Mult *int1 int2*)
: Return the result of (*int1***int2*).

(Div *int1 int2*)
: Return the result of (*int1*/*int2*).

(StrCopy {*str*} *int1 int2*)
: Return the string whom is between position int1 and int2. For example, (StrCopy {Hello} 1 2) returns {He}

(LaunchScript {*str*})
: This function launches the script named str and returns an identification number. This number is necessary to use the functions SendToScript and ReceiveFromScript. The string str contains the script name and some arguments.

(GetScriptArgument {*int*})
: This function returns the argument script used in the function LaunchScript. If int is equal to zero, GetScriptArgument returns the name of the script.

(GetScriptFather)
: This function returns the identification number of the script father.

(ReceivFromScript {*int*})
: This function returns the message sent by the script numbered int.

(RemainderOfDiv {*int1 int2*}): t
: This function returns the remainder of the division (*int1*/*int2*).

(GetTime)
: This function returns the time in seconds.

(GetPid)

This function returns the process id of the script.

(Gettext {*str*})

This function return the translation of *str* by using the locale catalog(s) defined with UseGettext.

(SendMsgAndGet {*comId*} {*cmd*} *bool*)

Sends the command *cmd* with identifier *comId* to an external program ready to communicate with the script using a protocol specific to FvwmScript. If *bool* is 0 FvwmScript does not wait for an answer from the external program. In this case the returned value is 1 if the message can be sent to the external program and 0 if this is not the case. If *bool* is 1, then FvwmScript waits for an answer from the external program and the return value is this answer (a line of no more than 32000 characters). If the communication fails, the returned value is 0. See the section **A COMMUNICATION PROTOCOL** for a description of the communication protocol used.

(Parse {*str*} *int*)

where *str* must be a string of the form:

```
      X1S1X2S2X3S3...SnXn
```

where the Xn are numbers containing four decimal digits and where Sn are strings of length exactly Xn. The returned value is the string S*int*. If*int* is out of range (e.g., >n) the returned v alue is the empty string. If *str* is not of the specified form, the return value is unpredictable (but empty in the average). This function is useful to handle strings returned by the SendMsgAndGet function.

(LastString)

This function returns the "current working string" for the Key instruction and the SendString command (see the **COMMANDS** section). At startup this string is empty, but when a Key binding is detected (respectively, a SendString command is received), then this string is set to the string associated to the instruction (respectively, to the command).

CONDITIONAL LOOPS

There are three kinds of conditional loops. The instruction "If-Then-Else" has the following syntax:

```
If $ToDo=={Open xcalc} Then
 Do {Exec xcalc &}                  # List of instructions
Else
Begin
 Do {Exec killall xcalc &}    # List of instructions
 Do {Exec echo xcalc killed > /dev/console}
End
```

The second part "Else-Begin-End" is optional. If the loop contains only one instruction, Begin and End can be omitted. The instruction "While-Do" has the following syntax:

```
While $i<5 Do
Begin
 Set $i=(Add i 1)             # List of instructions
End
```

Two strings can be compared with "==" and two numbers can be compared with "<", "<=", "==", ">=", ">". The loop "For-Do-Begin-End" has the following syntax:

```
For $i=1 To 20 Do
Begin
 Do {Exec xcalc &}                  # List of instructions
End
```

COMMANDS

The following fvwm command may be executed at any time

SendToModule *ScriptName* SendString *id sig str*

it sends to any module with alias or name which matches *ScriptName* the string

SendString *id sig str*

When an FvwmScript receives such a message it sends to the Widget *id* the signal numbered *sig* and the string *str* can be obtained with the LastString function. Let us give an example. Say that you have a script MyScript with the widget:

```
Widget 50
Property
 Type PushButton
 Title {Quit}
 ...
Main
Case message of

  SingleClic:
  Begin
    Quit
  End

  1 :
  Begin
    Set $str = (LastString)
    If $str == {Quit} Then
      Quit
    Else
      ChangeTitle 33 $str
  End

End
```

Then the command

```
SendToModule MyScript SendString 50 1 str
```

forces MyScript to exit if str is equal to "Quit" and if not it changes the title of Widget 33 to str.

This command can be used to change the window title

SendToModule *ScriptName* ChangeWindowTitle *newTitle [oldTitle]*

it causes that any module with alias or name which matches *ScriptName* changes its associated window title to *newTitle*. The optional argument *oldTitle* makes sense when there are several instances of the same script. It permits one to avoid changing the name of all these instances by specifying the name of the window associated to the target script (see the example below).

```
+ I Module FvwmScript FvwmStorageSend "/dev/hda6"
+ I Wait FvwmStorageSend
+ I SendToModule FvwmStorageSend ChangeWindowTitle HDA6
+ I Module FvwmScript FvwmStorageSend "/dev/hda1"
+ I Wait FvwmStorageSend
+ I SendToModule FvwmStorageSend ChangeWindowTitle HDA1 FvwmStorageSend
```

Without the FvwmStorageSend argument in the last case, the SendToModule command would have changed to HDA1 the name of both instances of FvwmStorageSend.

EXAMPLES

You will find examples of scripts in the fvwm configuration directory.

FvwmScript-BellSetup, FvwmScript-KeyboardSetup, FvwmScript-PointerSetup and FvwmScript-ScreenSetup are a set of scripts that modify X settings. These scripts save preferences into a file named ˜/.xinit-fvwmrc (If you want to use another file name, give it as the first argument of the script). If you want to load these preferences at every startup, you have to include the line ".xinit-fvwmrc" in your .xinitrc (or .xsession) file before starting fvwm.

FvwmScript-BaseConfig modifies fvwm focus and paging mouse policy, window placement, opacity and other features of the move and resize commands, snap attraction and shading animation. This script saves preferences into a file named .FvwmBaseConfig in the user's data directory (i.e., $HOME/.fvwm or $FVWM_USERDIR if set). If you want to load these preferences at every startup you must add the line "Read .FvwmBaseConfig" in your fvwm configuration file. If you want to use another file name, give it as the first argument of the script. When you click on Ok or Apply an fvwm function that you may define named BaseConfigOkFunc or BaseConfigApplyFunc is called. This allows for reloading specific application styles that the script has destroyed (e.g., AddToFunc BaseConfigOkFunc I Read MyAppStyle).

FvwmScript-Buttons is a buttons panel which can replace FvwmButtons (this script supports popup menus and requires xload, xclock, FvwmPager, TkDesk). FvwmScript-Colorset allows you to edit your colorset. FvwmScript-Date allows you to set date and time. FvwmScript-FileBrowser is a file browser used by the other scripts. FvwmScript-Find is an elementary front-end to find. FvwmScript-Quit allows one to quit fvwm, restart fvwm or some other window manager, or shut down and reboot the computer. FvwmScript-ScreenDump is a screen dumper. FvwmScript-WidgetDemo is a pure example script. See the next section for FvwmScript-ComExample.

A COMMUNICATION PROTOCOL

FvwmScript is a weak (but simple) programming language. If you need to deal with a lot of data and/or you need to use complex algorithms you should use an external program (in perl for example) and "send" the desired information to your FvwmScript script. The first approach is to use the GetOutput function. This is simple but you should rerun your external program each time you need information from it (and this may cause performances problems). The second approach is to use the SendMsgAndGet function which extends FvwmScript by using any programming language which can deal with named pipes (fifos). We describe this solution in this section. (A third approach is to use fvwm-themes-com from the fvwm-themes package, but in fact the SendMsgAndGet method is an implementation of fvwm-themes-com inside FvwmScript and this gives better performance).

Basically, you start an "external" program (the program for short) from your FvwmScript script (the script for short). This program runs in the background and you use the SendMsgAndGet function in your script to ask questions or to give instructions to the program. The program must strictly respect a certain communication protocol. First of all there is an identifier *comId* for the communication, it should contain the process id of the script for a good implementation of the protocol (use the GetPid function and pass the *comId* via an option to the program). The protocol uses two fifos, in the fvwm user directory, named: .tmp-com-in-*comId* and .tmp-com-out-*comId*. The program should create and listen on the .tmp-com-in-*comId* fifo. Then, when FvwmScript executes a function of the form:

Set $answer = (SendMsgAndGet {*comId*} {*cmd*} *bool*)

FvwmScript writes the *cmd* on this fifo. This way the program can read the *cmd* and can execute the appropriate action (it should remove the in fifo to support multi-communications). If *bool* is 0, FvwmScript does not wait for an answer from the program and return 1 if the previous actions succeed and 0 if they failed (then the program should "go back" to the in fifo). If *bool* is 1, then FvwmScript waits (20 sec) for an answer from the program and in turn returns the answer to the script (note that *bool* is not passed to the program as it must know which commands need an answer). To answer, the program creates the .tmp-com-out-*comId* fifo and writes the answer on it. The program should wait until FvwmScript reads the answer and then it should remove the out fifo and go back to the in fifo. The answer should consist of one line of no more than 32000 characters (take a look at the Parse function to handle multiple lines as one line).

A simple way to understand this protocol and to write scripts and programs that use it is to take a look at the (not useful) example FvwmScript-ComExample and fvwm-script-ComExample.pl (that can found in the fvwm data directory). Moreover, this implementation of the protocol solves questions as: What to do if the script exits for a bad reason? What to do if the program exits for a bad reason? ...etc.

BUGS

FvwmScript crashes if widgets are accessed that have not been defined.

AUTHOR

Frederic Cordier (cordie97@cui.unige.ch or f-cord96@univ-lyon1.fr).

CONTRIBUTOR

Eddy J. Gurney (eddy@gizmo.aa.ans.net).