Active Window software

Paul Spurgeon
1 August 2006

Overview

Active Window monitors a data from either USB IR Sensors, or a PLC connected via a serial port, and uses received data to control the PC's display. Any number of named states can be defined, each of which is displayed by an external viewer application. This release has been tested with these viewers (see Viewers section below):

PowerPoint slide shows
video files played by Windows Media Player
html files displayed by Internet Explorer
Flash swf files played by SWF Opener

For each state, an action map is defined, mapping codes into a new state and associated actions. Actions can be keyboard or mouse events, a request to launch a new application (and close the current one), IR Sensor commands, or script functions.

One display is defined as the start state. Transition from one display screen to another (from one state to another) occurs when the user generates a commands (eg using IR Sensors or switches). In addition, transitions can optionally occur when a state times out, or when the system is inactive for a long period (screensaver state).

Planning on paper is a good idea: it is very easy to change to the wrong state and/or give the wrong action.

When launched, Active Window runs minimised and displays the start state.

To exit, close the current display, then close Active Window.

When using PLC hardware, successfully executed global commands are echoed to the PLC via the serial port with CRLF appended - failed commands echo "failed" + CRLF.

Active Window monitors the system's http site, as defined by the SysID and password, for presentation script updates. If available, a Windows Script File (wsf) is downloaded and executed. Any wsf script can be used - support is provided for creating folders, downloading files using http, and copying local files. When all wsf commands have successfully executed, Active Window switches to the new presentation script when the system is idle.

Log files are periodically uploaded to the system's http log file area for on-line analysis.

1. Buttons

Named buttons are defined by selecting an IR Sensor ID and defining an activation zone using maximum and minimum ADC values. When zone monitoring is enabled, firmware in each IR Sensor monitors for transitions between zones, and sends zone change events to the PC. These events are dispatched to all groups for processing.

Depending on the group state, an IR Sensor event can cause buttons to be pressed or released. If several zones are defined for the same physical IR Sensor, software will ensure only one button is pressed at a time, and others are automatically released.

When a button is pressed or released, a press or release feedback event is generated.

When a button is pressed, a 100 mS timer counts down the button's pre-click delay time. If the counter reaches 0, a button click event is generated, sending the button's code to the Active Window state machine. When a click event is sent, a start click feedback event is also generated.

After a click event, a timer counts down the post-click delay time; when it becomes zero, an end click feedback event is generated. The button remains pressed until the IR Sensor returns a release state, or press in another zone.

Button definition:

Name	any unique name
ID	physical IR Sensor ID 1-255
Zone	IR Sensor zone index, 1-5
Min	minimum ADC value for zone
Max	maximum ADC value for zone
Code	Active Window code sent on click event
Pre-click time	delay time from press to click (x 100 mS)
Post-click time	delay time from click to end click (x 100 mS)
Press feedback	feedback action on button press
Release feedback	feedback action on button release
Start click feedback	feedback action on start click event
End click feedback	feedback action on end click event

2. Feedback

Feedback for each button is provided by RGB LED boards connected to IR Sensor boards. Each feedback item is a message with an IR Board ID where the message will be sent when a feedback event is triggered.

Typically each button press event sets an RGB LED to a suitable feedback colour, a start click event changes the colour, an end click event changes the LED back to the original colour (showing the button is still pressed), and a release event turns the LED off.

3. Groups

All buttons are assigned to a named group with a button interlock mechanism, and a flag to indicate if group activity (ie button press events) resets the system idle counter.

The interlock mechanism is one of:

Single: only one button can be pressed at a time, and the button must be released before another in the group can be pressed (wait for release before accepting a new press event);

Any: any button in a group can be pressed at any time, independently of others in the group;

Last: the last button press is accepted; all others in the group are released.

A group is either always active (ie monitoring button press events), or only active when one or more groups monitored by this group become idle (ie no buttons pressed). When a group becomes inactive (because of a button press event in one of the monitored groups), any pressed buttons are automatically released.

When a group brings the system out of idle (only those groups with the reset idle flag set), the user log file records a “Start” user interaction session event, with the associated group code. Group codes are reserved as:

1000 – 1999	panel button groups
2000 – 2999	window watcher button groups
3000 – 3999	walk by button groups

4. Idle

The system monitors active groups with the reset idle flag set. If none of these groups is active (ie no press events are being processed) before the idle time passes, the system sends a idle code to the Active Window state machine. The idle delay & code are specified in Buttons.ini:

```
[Idle]
Timeout=100  //number of 100 mS ticks before timeout (100 = 10 seconds)
Code=84      //Active Window code sent on idle event
```

5. State machine events & actions

Events are sent as a result of button clicks or when the button controller detects an idle state, and are sent as a single byte character code (compatible with earlier versions, including PLC). Actions are selected using the code: first the global actions defined in the [GLOBAL] section of ActiveWindow.ini are checked; if none match the event code, actions for the current state (as defined in the executing presentation script) are checked. Global actions have a different format to script actions.

Global actions are one of:

Action	Description
idle	system enters idle state
home	Presentation restarts (from [START] Run=)
exit	system app closes
shutdown	app & PC closes down
powerdown	app & PC closes down; PC powers down (if supported by hardware)
reboot	app & PC closes down & reboots
update	app runs new presentation (from [UPDATE] Run=)

For each state, actions are assigned to command codes. Optional actions which will be executed on entering and leaving a state may be specified. And a state timeout may also be specified. Any number of script commands, starting with & separated by colons (':'), can be specified – they are executed sequentially, left to right.

For example:

```
[A2]
ENTER=:(44,50,122,1,3,0):FX FLASH.LOOP
LEAVE=:(44,50,122,1,1,0):FX FLASH.STOP
49=:"{LEFT}":A1
50= :<100,100,1>:A3
51=:SCRIPT BT(1000 [rækkehus.ppt]:B1 1500 [gæt.ppt]:C1 [valg.ppt/n3]:A3)
WAIT=50,:[gæt.ppt/n3]:C3
```

defines these actions for state A2:

on entering this state, turn a relay on & run the flash special effect in a loop
on leaving this state, turn the same relay off & stop the flash special effect
on code 49 ('1'), send cursor key left & goto state A1
on code 50 ('2'), click mouse left button at (100,100) & goto state A3
on code 51 ('3'), run script command BT (before time)
after 5 seconds idle, show PowerPoint presentation page 3 & goto state C3

Whenever the an action list causes the system to change state, actions for leaving the old state are executed, then actions for entering the new one, before continuing to execute the current action list.

Applications are launched using file names (no spaces allowed) and optional command line switches (using '/' or '-' separators) enclosed in square brackets '[' and ']' with no spaces or colons, for example:

```
[gæt.ppt/n4]     // run a PowerPoint presentation and go to page 4
[more\m1.avi]    // run a video from a sub-directory in the current presentation
[rigtig.htm]     // show an html page
```

Application paths for each file extension, together with global switches and close down mechanism are in Active Window.ini.

Key press events are specified using the SendKeys format (see SendKeys.txt), in double quotes, for example:

```
"4{ENTER}"    // send character '4', then send the keyboard 'enter' key code
"{RIGHT}"     // send the keyboard cursor right arrow key code
"{SPACE}"     // send the character generated by the keyboard spacebar
```

Mouse actions are specified as screen coordinate pairs & an optional click, in angled brackets '<' & '>' with no spaces, for example:

```
<x,y>      // move mouse to x,y
<x,y,1>    // move to x,y and click left button
<x,y,2>    // move to x,y and click right button
<x,y,3>    // move to x,y and click middle button
```

IR Sensor commands are an ID & list of byte codes (see "IR Sensor Hardware & Firmware.doc" for a complete description of byte code commands) in round brackets '(' & ')' with no spaces. For example:

```
(42,49,1,0,0,0)        // send "49,1,0,0,0" to Sensor ID 42 = turn red LED on
(43,50,126,1,2,0)      // send "50,126,1,2,0) to Sensor ID 43 = send I2C
                          command "1,2,0" to RGB board (126) = set RGB
                       LEDs to green
(44,50,122,1,3,0)      // send "50,122,1,3,0" to Sensor ID 44 = relay on
```

Script commands start with the keyword SCRIPT, followed by a script function and a list of parameters in round brackets, separated by spaces. Available script functions are:

```
SCRIPT BT(<HHHH> acts <HHHH> acts …. ELSE acts )
SCRIPT BD(<YYYYMMDD> acts < YYYYMMDD > acts …. ELSE acts)
```

BT (before time) and BD (before date) select a action lists (acts) using the current time/date. If the current time/date is before the first one, the first action list is executed; if before the second, the second list is executed, and so on. If the current time/date is not before any listed, the ELSE list is executed. For example:

```
SCRIPT BT(1000 B1:[valg.ppt/n1] 1500 C1:[valg.ppt/n2] ELSE A3:[valg.ppt/n3])
```

if the time is before 1000, show page 1 & goto state B1, if before 1500 show page 2 & goto C1, otherwise show page 3 of the PowerPoint slide show & goto A3.

Macros can be defined in presentation scripts for frequently used or complex actions, and to improve readability. Macro commands are single words (no spaces or punctuation), ending in an exclamation mark ‘!’. For example:

```
[MACROS]
ON!=(44,50,122,1,3,0)
OFF!=(44,50,122,1,1,0)
AM!=A1:[valg.ppt/n2]:ON!
PM!=A2:[valg.ppt/n3]:ON!

[SCREENSAVER]
WAIT=600,SCRIPT BT(0630 A3:OFF! 1200 AM! 2245 PM! ELSE A3:OFF!)
```

Turns a relay (maybe controlling the display screen) on sensor ID 44 off at the end of the day, and on in the morning. Also selects between a morning and afternoon presentation.

6. FX: special effects

Any number of special effects may be defined in a presentation. Special effects run on timer ticks after scripts have finished execution. Special effects are named with the prefix “FX” followed by a space. Each special effect consists of an initialisation part (INIT), a sequence of actions (SEQ), and a stop part (STOP): any script actions can be used. In addition, the WAIT(<ticks>) command can be used in the list of SEQ actions. Special effect commands can be one of the following:

Command	**Effect**
.INIT	run FX INIT actions
.RUN	run SEQ actions once
.LOOP	run SEQ actions forever (until STOP)
.REPEAT(<times>)	repeat SEQ actions a number of times (or until STOP)
.STOP	stop FX RUN/LOOP/REPEAT & run FX STOP actions

Some examples:

```
[FX FLASH]
SEQ=:ON!:WAIT(2):OFF!:WAIT(2)
STOP=:OFF!

[A1]
Enter=:FX FLASH.REPEAT(25)
Leave=:FX FLASH.STOP
```

On entering state A1, start the flash special effect in a repeat loop. On each 0.1 second tick, the system will run the flash SEQ actions, so first the LED is turned on, then the system waits 0.2 seconds, turns the LED off, waits another 0.2 seconds, and repeat this sequence 25 times or until FLASH.STOP (executed on leaving this state).

```
[FX DEMO]
INIT=:[rækkehus.ppt/n2]
SEQ=:B2:WAIT(5):B3:WAIT(5)
STOP=:A1

[SCREENSAVER]
WAIT=50,:FX DEMO.INIT:FX DEMO.LOOP
Leave=:FX DEMO.STOP:[valg.ppt]:A1
```

The screensaver waits for 5 seconds, then initialises the demo FX by showing page 2 of a PowerPoint slide show, and starts a loop forever executing the demo SEQ, which goes to state B2 (the Enter action of B2 shows page 2), waits 0.5 seconds, goes to state B3 (Enter action shows page 3), waits another 0.5 seconds, and repeats this sequence forever, or until DEMO.STOP. The Leave action for the screensaver is executed when the system leaves the screensaver state (when a button in a reset-idle group is activated): the action stops the loop and goes state A1.

Special effects (especially LOOP & long REPEATs) run continually until STOPped, regardless of the state.

To avoid conflict & confusion, it might be a good idea to not change state within special effects – enter/leave scripts in each state can conflict with the state changes in FX scripts! Instead, use special effects to control relays & RGB LEDs.

7. State machine timer actions

Various actions are initiated by timers:

Timer	Loaded from	Action
state	presentation script: [<current state>] Wait=	runs wait action after wait time in a state
screen saver	presentation script: [SCREENSAVER] Wait= Active Window.ini: [OPTIONS] Autorun=	after idle time, checks autorun script & runs if changed; otherwise runs screensaver wait action
upload log	ActiveWindow.ini [LOG] Ticks=	uploads all logs via http
update presentation	ActiveWindow.ini [UPDATE] Ticks=	downloads new presentations (if available)
update	presentation script: [UPDATE] Wait= Update.ini: [UPDATE] Run=	runs new presentation (from [UPDATE] Run=)

```
[G1]
ENTER=(42,49,1,0,0,0)          // turn red LED on sensor ID 42 on entering state
WAIT=10,(42,49,0,0,0,0):G2     //after 1 second, turn the off & goto state G2.

[SCREENSAVER]
WAIT=300,[valg.ppt/n1]:A1      // after 30 seconds of idle (no panel type sensor
                               events), show page 1 of a PowerPoint presentation
                               and goto state A1.

[LOG]
Ticks=60        // try to upload log files once a minute.

[UPDATE]
Ticks=600       // try to download a new presentation once every 10 minutes.


[OPTIONS]
Autorun=SCRIPT BD(20061001 [demo.txt] 20061116 [xmax.txt] 20061225
[newyear.txt] ELSE [demo2007.txt])
```

8. ActiveWindow.ini

```
[SYSTEM]
Hardware=PLC        // for backwards compatibility = PLC; IR Sensor hardware = USB
Buttons=Buttons.ini  // selects USB button configuration for this presentation

[FOLDERS]
Current=Presentation0000

[OPTIONS]
UserLog=71          //user log serialization
SystemLog=71        //system log serialization
Autorun=Demo.txt    //start up presentation script
Show=2              //start Active Window maximised (0= hidden, 1 = normal)

[USER]
SysID=1
Key=abcdefgh

[UPDATE]
Host=http://www.activewindow.dk/information
Proxy=
Asp=upd.asp
Cmd=dlf
File=Update.wsf
Ticks=120    //update request interval (seconds): 0 = no update
Abort=60     //interval before aborting update script (seconds)

[LOG]
Host=http://www.activewindow.dk/information
Proxy=
Asp=log.asp
Cmd=add
Ticks=120    //log upload interval (seconds): 0 = no upload
```

9. Log file upload mechanism

At the log tick interval ([LOG] Ticks=) the app attempts to upload old user and system log files (*.log) by sending an HTTP header formed from the [LOG] section in ActiveWindow.ini. If the upload succeeds, local log files are deleted. If it fails the app tries again later. When there are no old log files to upload, the log file indices are incremented, and the old log files will be uploaded after the next tick interval.

Log file indices are also incremented when the log file size grows over approximately 10k, to ensure log files do not grow too large for HTTP uploads.

10. Presentation update mechanism

Periodically the app requests an update script (update.wsf) from an http site using the url formed by data in the [HTTP} section of ActiveWindow.ini: if an update script is successfully downloaded (to the users temp directory), the app copies script helper files & writes a temporary file (update.ini), then executes the wsf script asynchronously. The wsf script reads from and writes to this ini file, and eventually exits, either returning success or an error code. If the script completes successfully the new presentation is displayed once the system goes idle. If it fails, the update is aborted, and the process runs again later.

Just before running the update.wsf script, update.ini looks like this:

```
[FOLDERS]
Root=C:\Documents and Settings\Paul\My Documents\My Active Window\My Presentations
Install=C:\Program Files\Active Window ApS\Active Window
Current=Presentation0000

[UPDATE]
State=0         //current wsf script state
Run=            //presentation to run
```

Root is the path for installed and added presentation folders; Install is the path for ActiveWindow.exe and ActiveWindow.ini. The current presentation folder is also passed to the wsf script – if this is the same as the folder to be installed by the script, it will exit without performing any actions.

Update scripts are usually implemented as a state machine, with initial state = 0. When the final state has been reached, or an update error occurs, the script must return success or an error value (0 = success, 100 = no need for an update). For success, the update script will also write the name of the new presentation folder, the filename of the presentation to run, and optionally the folder to delete. After success the update.ini file looks like this:

```
[FOLDERS]
Root=C:\Documents and Settings\Paul\My Documents\My Active Window\My Presentations
Install=C:\Program Files\Active Window ApS\Active Window
Current=Presentation0000
New=Presentation0001
Remove=Presentation0000

[UPDATE]
State=10
Run=Demo.txt
```

The app switches to the new presentation folder and runs Demo.txt.

For details of wsf file commands, see the header in fso.vbs, which contains support routines for update scripts.

11. PC reset relay

A relay board may be connected to one of the IR Sensors and configured as a PC reset: by connecting the relay contacts to the PC reset button, a software crash will close the reset relay (for 0.5 seconds) and reboot the PC.

The reset relay is configured in Buttons.ini:

```
[Reset]
ID=44          //ID of sensor connected to reset relay
Ping=5         //interval between pings from PC to reset relay
Ticks=15       //timeout before reset relay is activated if no pings received
```

Both ping interval and ticks are in seconds. For this example, the PC sends a ping to sensor ID 44 every 5 seconds. The sensor will activate the reset relay if no ping is received after 15 seconds. Ticks should be greater than the ping interval! The reset relay must be set to address 122 (0x7a) – ie the PCB jumper should not be fitted.