Command-Line SLX

Introduction

SLX is designed to be launched easily from a variety of contexts. It can be launched from within another program, from a command script, or by directly typing a command within a command prompt window. We refer to all such launches of SLX as command line invocations. This memo describes such invocations of SLX.

The same executable file is used for command-line execution and for fully interactive execution. For 32-bit SLX, the executable is se32.exe, and for 64-bit SLX, se64.exe. All the examples in this memo illustrate the use of 32-bit SLX, but apply equally to 64-bit SLX. When SLX is invoked by clicking on an icon or clicking on an SLX file name, the full, interactive SLX development environment is launched. In contrast, command line launches generally run SLX in "batch" mode.

By using the "/silent" command line option, you can force SLX to write any error messages to a log file and quit, rather than popping up a dialog and waiting for input from a non-existent interactive user

Launching SLX

The most common ways of performing command line launches of SLX are (1) calling the Windows LoadProcess() API function (usually from C++ or Visual Basic), (2) invoking SLX from a command script (.bat file), and (3) launching additional copies of SLX from within an SLX program, using SLX's LaunchProgram() procedure.

Setting up Directories

It is your responsibility to specify the directory in which SLX starts. For example, SLX's LaunchProgram() procedure includes an argument that specifies the name of the directory to be used.

Waiting for Completion

It is your responsibility to wait for completion of a command line invocation. If you launch SLX using the LoadProcess() Windows API function, you can use the WaitForSingleObject() API function to wait for completion. In a batch script, you can use a "start/wait" command, rather than just invoking se32.exe. If you launch SLX from an SLX program, the LaunchProgram() procedure includes a synchronization argument that is specified as SYNC or ASYNC.

Testing Return Codes

SLX always returns a status code. You can use this code to react to unexpected events. If your program terminates with an exit(*xpr*) call, the value of *xpr* is returned. If you program simply runs to completion (no exit() call), a value of zero is returned.

The following error return codes are issued by SLX:

Code Name	Value	Explanation
EXIT_KEY_INTERROGATION	-10001	Unable to find a security key with SLX permission
EXIT_BAD_OPTION	-10002	The command line includes an invalid option.
EXIT_BAD_SOURCE_FILE	-10003	The source file cannot be found.
EXIT_BAD_OUTPUT_FILE	-10004	The specified/implied output file cannot be written.
EXIT_COMPILATION_ERROR	-10005	The program contains compile-time errors.
EXIT_RUNTIME_ERROR	-10006	The program has terminated with a run-time error.
EXIT_GENRTS_FAILURE	-10007	The /genrts option failed.

Command Line Format

Assuming that SLX has been installed in c:\wolverine, the form of a command line is as follows:

C:\wolverine\slx\se32 [/option1 ... /optionN] filename [main procedure argument…]

All command line options start with a "/" character. The first item after se32 in the command line that does not start with a "/" is interpreted as the name of the file to be run. File names that contain embedded blanks must be enclosed in quotation marks ("…"). The file to be run can be an SLX source file (.slx suffix) or a run-time SLX file (.rts suffix). Tokens to the right of the file name are passed as arguments to the program's main procedure.

Command Line Options

Option	Action
/background	Run at background priority
/capture	Capture any C++ output as if it were SLX "print" output.
/fullscreen	Run SLX in a full-screen window
/genrts [fn2] fn1	Compile fn1,slx and generate an RTS file named fn2.rts (no execution) If a single file name is specified, by default fn2 = fn1
/ignore #name	Ignore #defines for “#name”
/imports	List imported files
/inlineinit	Process inline variable initialization in C++ style
/keyretrycount nnn	Set the retry count for failed security key interrogations. The default retry count is 3. When a nonzero retry count is specified, SLX pauses five seconds between retries. This option is useful in contexts where many runs are launched nearly simultaneously, and there is contention for the key server.
/lis	Write "print" output to <source file>.lis
/localscopes	C++-style local variables in local scopes, exceptions: declarations in “actions” and “initial” properties are class-wide
/networkretry	Assume the security key is a network key. If the key cannot be found at a previously discovered network address, initiate a new search.
/noicon	Suppress taskbar SLX icon
/norun	Suppress execution
/nowarn	Suppress compiler warnings
/noxwarn	Suppress execution warnings
/output fname	Place "print" output in "fname"
/quiet	Suppress status messages
/rtf	Write "print" output to <source file>.rtf
/rts	Run a Run-Time SLX program (.RTS file)
/run	Necessary to force execution *only* if no other options are given
/screen	Run SLX in a default-sized window
/silent	Run with no user interaction, suppressing all but disastrous messages
/SLX2	Enable SLX2 features
/stdout	Write ASCII output that can be redirected

/strictlocalscopes	Strict C++-style local scopes (no exceptions for “actions” and “initial” properties)
/wait	Wait for program to complete (synchronous execution)
/#xyz	Define symbol "xyz" for use with #ifdef and #ifndef

Interactive vs. Non-Interactive Runs

If *no* command line options are specified, behavior depends on whether the source file is a .slx file or a .rts file. If the source file is a .rts file, SLX assumes that execution is required, and it loads and executes the file. If the source file is a .slx file, SLX brings up the SLX IDE with the source code of the file displayed in a window. The program is neither compiled nor executed. Note that a command line with no options that specifies a .slx file is indistinguishable (to SLX) from double-clicking on the .slx file from within the Windows Explorer (assuming that se32 or se64 has been registered as the program for executing .slx files).

If you want to *run* a .slx file from a command line invocation, and you have no need to specify any other command line options, you *must* supply the /run option to force execution.

By default, SLX assumes that an interactive user is present. If a command line invocation is made in a context in which there is no interactive user, the /silent option must be used to assure that all output is written to files and no requests for interactive are made.

Controlling Output Content and Destination

By default, command line execution of a program places all "print" output in the file named <filename>.rtf. To place output in <filename>.lis (unformatted ASCII) file, use the /lis option.

By default, the full range of SLX output is generated. To reduce the amount of output "noise," you can use the /quiet option. This option suppresses the production of messages such as "Execution begins" and end-of-run statistics.

The /silent option minimizes SLX output noise and, more importantly, it affects error handling, as described below.

If output is to be routed to the screen, you must use the /screen or /fullscreen option.

Alternate Output Files

Under certain circumstances, SLX may be unable to open the file that would normally be used for program output. For example, if the command line specifies execution of an improperly-formed file name, SLX cannot construct <filename>.rtf or <filename>.lis as a destination file name.

SLX chooses destinations in the following order:

1. User-specified "/output" destination
2. <filename>.rtf or <filename>.lis
3. <filename>.log
4. SLX.log

The latter two options are used as a last resort when severe errors occur, e.g., failure to recognize a Wolverine security key, hardware exceptions, etc.

Dealing with Errors

By default, SLX assumes that an interactive user is present to react to error conditions. When an error occurs, the SLX IDE pops up, displaying error information. If command line invocations are run in a context in which no interactive user is present, you should use the "/silent" option. This option forces SLX to write all error output to a file and terminate with a return code reflecting the nature of the error.

Command Line Examples

The example shown below illustrates the passing of command-line arguments to an SLX main procedure. The number of command-line arguments is passed in argc, and the arguments themselves are passed via argv, a 1-dimensional array of SLX strings. The following rules apply:

1. The minimum value of argc is 1.
2. The indices for argv run from 0…argc-1.
3. The fully-qualified name of the program file being run is passed as argv[0].
4. The first argument to main, if any, is passed as argv[1].

```
procedure main(int argc, string(*) argv[*])
    {
    int  i;
    print (argc    )     "args: _\n\n";

    for (i = 0; i < argc; i++)
        print(i, argv[i])                "arg _: _\n";
    }
```

Sample command lines follow:

C:\wolverine\slx\se32 /run myprog	Uses all defaults. The program executes in an invisible, minimized window. "print" output is written to myprog,rtf. Any errors result in the SLX IDE popping up.
C:\wolverine\slx\se32 /lis myprog	Same as above, but output is written to myprog.lis
C:\wolverine\slx\se32 /noxwarn /quiet myprog	Same as above, except execution warnings and status messages suppressed.
C:\wolverine\slx\se32 /silent myprog	Same as above; however, all output (even for severe errors) is routed to files, and the SLX IDE *never* pops up.
C:\wolverine\slx\se32 /screen myprog myopt1	Output written to the screen, including status messages. "myopt1" is passed to main program as argv[1].
start/wait C:\wolverine\slx\se32 …	Synchronous execution (wait for completion).