SLX-to-EXE

Introduction

This memo describes how to package an SLX program as a self-contained, dongle-free EXE file. The process is quite simple:

1. Compile the SLX program.
2. Click on "Package App as an EXE File" in the File menu.
3. Supply the name of the EXE file to be created in the standard "write file" dialog that pops up.
4. Supply any command line options (see below) to be hard-wired into the EXE file.

Source code for the SLX model is encrypted in the EXE file, and it is hidden from the user. If execution errors occur, the line(s) of source code in which the error occurs are reconstituted and highlighted, allowing the model builder to see where things went wrong. These are the *only* circumstances under which source code is visible.

A model can be run in a standard SLX window that provides limited run control menu items, but lacks all other model development items, or it can be run "silently" with no window, presumably placing all its output in files.

Launching an Application

An SLX application packaged as an EXE file is launched just like any other Windows application. It can be launched from a command line in a command prompt session, from a BAT file, or from within another under program, e.g., a "front end," using an API function such as ShellExecute() or LoadProcess(). The program uses the fully-qualified name of the EXE file its default directory, which affects how any input files are located and where output files are written.

Command-line options can also be specified in the invocation. If so, they are appended to the right of any options built into the EXE file.

All SLX-enforced command-line options begin with a "/". The first command line option that does *not* start with a "/" is interpreted as the first argument to be passed to the SLX main program, along with any succeeding options. SLX uses the C/C++ "argc" and "argv" convention for main program arguments; i.e., argv[0] is the name of the EXE file being run, and argv[1] is the first (if any) non-SLX command-line option.

Waiting for Completion

It is your responsibility to wait for completion of a command line invocation by using the /wait command-line option. If you do not use this option, the EXE will execute asynchronously when invoked from another program or from a BAT file script.

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 your 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_BAD_OPTION	-10002	The command line includes an invalid option.
EXIT_BAD_OUTPUT_FILE	-10004	The specified/implied output file cannot be written.
EXIT_COMPILATION_ERROR	-10005	The program contains compile-time errors. Errors of this type can occur for programs containing data-driven macro or statement expansions.
EXIT_RUNTIME_ERROR	-10006	The program has terminated with a run-time error.

SLX 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 fname	Generate an RTS file named fname.rts (no execution)
/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

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 <EXE filename>.rtf. To place output in <EXE 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 <EXE filename>.rtf or <EXE filename>.lis as a destination file name.

SLX chooses destinations in the following order:

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

The latter two options are used as a last resort when severe errors occur, e.g., hardware exceptions.

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 invocation of an SLX program that has been compiled and bound into myprog.exe.

```
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:

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.
Myprog	/lis	Same as above, but output is written to myprog.lis
Myprog	/noxwarn /quiet	Same as above, except execution warnings and status messages suppressed.
Myprog	/silent myprog	Same as above; however, all output (even for severe errors) is routed to files, and the SLX IDE *never* pops up.
Myprog	/screen myopt1	Output written to the screen, including status messages. “myopt1” is passed to main program as argv[1].
Myprog	/wait	Synchronous execution (wait for completion).

Avast Internet Security

Those of you who are using Avast Internet Security may experience random complaints from Avast during the EXE generation process. Avast sometimes thinks that evil intent lurks inside the EXEs SLX generates. If you encounter this problem, you can try adding the EXE file to Avast’s local exemption list and regenerating the EXE file. If this fails, you’ll have to temporarily turn off Avast’s file protection feature. Note that Avast’s file protection system has a long history of generating false positives. If you perform an Avast file scan on an SLX-generated EXE file it has previously complained about, the scan will indicate that the file is OK. Go figure!