

sphinxcompiler Reference

mpark@sphinxcompiler.com

Reference

Core programs

- [sphinx.bin](#)
- [lex.bin](#)
- [codegen.bin](#)
- [link.bin](#)

Editor

- [ed.bin](#)

Utilities

- [echo](#)
- [cogcheck](#)
- [dir](#)
- [copy](#)
- [del](#)

Device drivers

- [sxf](#)
- [sxkb](#)
- [sxtv](#)

Device interfaces

- [isxfs](#)
- [isxkb](#)
- [isxtv](#)
- [sxfile](#)

PC utilities

- [put](#)
- [get](#)

Core programs

sphinx.bin

Sphinx.bin installs the Sphinx drivers for the keyboard, TV, and SD card. It also acts as a simple command-line shell.

A command line can take any of the following forms:

- *filename arg₁ arg₂ ... arg_n*
- **run** *filename arg₁ arg₂ ... arg_n*
- **c** *filename arg₁ arg₂ ... arg_n*
- **cl** *filename arg₁ arg₂ ... arg_n*

The first form loads and executes the file named *filename*. If *filename* ends with

".bin" or ".eep", sphinx.bin will execute the file of that name (if it exists, of course). Otherwise, sphinx.bin will attempt to execute *filename*.bin and, if that fails, *filename*.eep.

The second form, the **run** command, also loads and runs *filename*. The difference between these two forms of command is that the first one leaves the Sphinx device driver cogs running, while the second stops all cogs (it effectively resets the Propeller). Use the first form to execute Sphinx-aware programs; use the **run** command to execute regular programs.

The **c** command compiles *filename*.spn and generates *filename*.sob (assuming no errors). This command runs [lex.bin](#) and [codegen.bin](#), passing any command-line arguments to both programs (see below).

The **cl** command compiles and links *filename*.spn, generating *filename*.sob and *filename*.bin (again, assuming no errors). This command runs [lex.bin](#), [codegen.bin](#), and [link.bin](#), passing its command-line arguments to all three programs.

Sphinx.bin should be stored in EEPROM so that the Propeller boots into Sphinx. Sphinx.bin should also exist on the SD card so that Sphinx-aware programs can, once they've finished their work, load and execute sphinx.bin.

Command-line arguments

Sphinx.bin stores arguments as ASCII strings in a file named "args.d8a". A program launched from Sphinx can read the argument strings from that file. The first byte of the file is the number of arguments; it is followed by that number of strings, each null-terminated.

As noted earlier, the **c** and **cl** commands run several programs and each program receives the same arguments. This is because they do not modify args.d8a before executing the next program in the sequence. Each program pays attention to the arguments it understands and ignores any that it does not.

lex.bin

Lex.bin is the first phase of the compiler. It takes a .spn file as input, tokenizes it, and writes the result to a .tok file. On successful completion, lex.bin can automatically run [codegen.bin](#), the next compiler phase.

Usage:

```
lex filename [options]
```

Options:

- `/c` -- run `codegen.bin` automatically on successful completion.
- `/l` -- run `codegen.bin` automatically on successful completion and tell it to run [link.bin](#).
- `/v n` -- set verbosity to 0, 1, 2, or higher (default is 0). The higher the verbosity, the more messages `lex.bin` prints as it runs.

Example:

```
lex tv_text /c
```

This command tokenizes `tv_text.spn`, producing `tv_text.tok`, and then runs `codegen.bin`. Note that you don't have to add `".spn"` to the input filename. `Lex.bin` will add it automatically.

You should never have to run `lex.bin` directly; instead, use the convenient `c` command (provided by [sphinx.bin](#)) which is equivalent to running `lex` with the `/c` option.

codegen.bin

`Codegen.bin` is the second phase of the compiler. It takes a `.tok` file generated by [lex.bin](#) and compiles it to a `.sob` file. On successful completion, `codegen.bin` can automatically run [link.bin](#) (of course, linking should only be done on the top object, not child objects, so don't use the `/l` option too freely).

Typically you will never run `codegen.bin` directly. Instead you will use the `c` and `cl` commands (provided by [sphinx.bin](#)) which run `codegen.bin` for you.

Usage:

```
codegen filename [options]
```

Options:

- `/l` -- run `link.bin` automatically on successful completion.
- `/s n` -- set stack size to `n` bytes (default is 2300). If `codegen.bin` fails because of stack overflow, run it again with a larger stack. `n` must be a multiple of 4.
- `/t n` -- set symbol table size to `n` bytes (default is 6000). If `codegen.bin` fails because the symbol table becomes full, run it again with a larger symbol table. `n` must be a multiple of 4.
- `/v n` -- set verbosity to 0, 1, 2, or higher (default is 0). The higher the

verbosity, the more messages codegen.bin prints as it runs.

Example:

```
codegen howdy /v 3 /t 7000
```

This command compiles howdy.tok and produces howdy.sob. While running, codegen.bin prints many informative(?) messages. The symbol table size is 7,000 bytes. Note that you do not have to add ".tok" to the input filename. Codegen.bin will add it automatically.

link.bin

Link.bin is the linker. It takes a top object .sob file and links it together with any child .sob files it requires, producing a .bin file.

Usage:

```
link filename [options]
```

Options:

- `/i` -- ignore out-of-date errors.
- `/v n` -- set verbosity to 0, 1, 2, or higher (default is 0). The higher the verbosity, the more messages link.bin prints as it runs.

Editor

ed.bin

Ed is a simple full-screen text editor. It displays 13 lines of 40 characters. There is no word-wrap; the whole screen scrolls sideways if the cursor is at the end of a long line. Ed is always in insert mode.

You invoke ed at the Sphinx prompt by typing "ed" followed optionally by a filename.

Ed understand the following key commands:

- ENTER -- Carriage return
- BKSP -- Delete to the left
- DEL -- Delete to the right
- arrow keys -- Cursor movement

- HOME -- Go to start of line
- CTRL-HOME -- Go to start of file
- END -- Go to end of line
- CTRL-END -- Go to end of file
- PGUP -- Page up
- PGDN -- Page down
- CTRL-O -- Open a file
- CTRL-S -- Save to a file
- CTRL-Q -- Save and quit to Sphinx

The last three commands give you the opportunity to enter or edit a filename. To accept the filename, hit ENTER. To cancel the file operation, hit ESC.

Implementation notes

Ed is a Sphinx-aware program that uses Sphinx keyboard and file I/O, but it uses its own video driver. Sxtv does not provide cursor positioning and other functions that ed needs, so ed contains a modified version of the Parallax tv_text object (which in turn contains the Parallax tv object). Ed uses sxtv.Disable to turn off Sphinx video while it's running. When ed is about to transfer control back to Sphinx, it calls sxtv.Enable to turn Sphinx video back on.

Utilities

cogcheck

Prints which cogs are active and which are free.

Usage:

```
cogcheck
```

copy

Copies a file.

Usage:

```
copy srcfile dstfile
```

Copies *srcfile* to *dstfile*. If overwriting *dstfile*, will ask for confirmation first.

del

Deletes files.

Usage:

```
del files [options]
```

Options:

- *Y* -- delete without asking for confirmation

files can contain wildcard characters ("?" and "*").

dir

Prints a directory of files.

Usage:

```
dir [files]
```

files can contain wildcard characters ("?" and "*"). If *files* is not specified, it defaults to "*.*".

echo

Prints its arguments.

Usage:

```
echo [arguments]
```

Not particularly useful, just an example of reading arguments from args.d8a.

Device drivers

sxfs

Sxfs is the Sphinx file system. It implements a barebones FAT16 file system on SD cards. It occupies three cogs and requires 139 longs of hub memory, most of which is a 512-byte metadata buffer. Sxfs supports multiple open files. Every open file requires 134 longs for a sector buffer and housekeeping variables. Sxfs can read a file, write a file, and load a file into hub memory and execute it.

Programs that want to use sxfs's functionality should use the sxfs interface object, [isxfs](#), or a higher level file object such as [sxfile](#). Only programs that need to install the sxfs driver, such as sphinx.bin, should use the sxfs object itself.

Spin interface method

- pub **Start**(*sdPin*) -- starts the three sxfs cogs if they are not already running. Returns true if the cogs needed to be started, false if they were already running.

Implementation notes

Communication to and from sxfs is done through a rendezvous of four longs. (The three sxfs cogs communicate with each other through their own private rendezvous locations.) The four longs are *command*, *param0*, *param1*, and *param2* (starting at hub address \$7fcc). To perform a file operation, set up the parameters and then set *command* to a command value. Wait until *command* becomes 0, indicating that the operation has completed, and examine *param0* for a return code. A negative return code indicates that an error has occurred.

Instead of waiting, one could arrange to continue doing something else while the file operation completes asynchronously. Sphinx does not take advantage of this capability.

The commands and their parameters are as follows:

command	param0	param1	param2	Description
"?"				Does nothing but acknowledge to indicate that sxfs cog is active.

"O"	<i>filestuff</i>	<i>filename</i>	<i>mode</i>	Open a file, populate <i>filestuff</i> fields. <i>mode</i> is "R" or "W".
"R"	<i>filestuff</i>	<i>buffer</i>	<i>num</i>	Reads <i>num</i> bytes into <i>buffer</i> from file specified by <i>filestuff</i> .
"W"	<i>filestuff</i>	<i>buffer</i>	<i>num</i>	Writes <i>num</i> bytes from <i>buffer</i> to file specified by <i>filestuff</i> .
"C"	<i>filestuff</i>			Closes file specified by <i>filestuff</i> .
"X"	<i>filestuff</i>	<i>execmode</i>	<i>cog</i>	Executes file. See below.

filestuff is a pointer to a 134-long area of hub memory. Each open file requires its own *filestuff* area.

The `execute` command reads 63 contiguous sectors from the SD card into hub memory. It does not read 64 sectors because the 64th sector would overwrite the rendezvous locations for the various Sphinx drivers. This could conceivably cause some files not to run properly.

Because `sxfs` assumes that a 32k file occupies 64 contiguous sectors on the SD card, the card must be formatted with a cluster size of at least 32k.

Of course, `.bin` files do not necessarily take up all 32k. `Sxfs` reads 63 sectors but then clears memory above the actual extent of the file.

If *execmode* is 0, `sxfs` starts up a Spin interpreter in cog *cog*. If *execmode* is non-0, `sxfs` shuts down all cogs except cog *cog*; it also resets the system clock if necessary.

The [isxfs](#) object takes care of all those details. The [sxfile](#) object additionally encapsulates a *filestuff* area.

One of the `sxfs` cogs runs a slightly modified version of `sdspiqasm.spin` by Tomas Rokicki.

sxkb

`Sxkb` is the Sphinx keyboard driver. It is functionally similar to [comboKeyboard](#) in the Object Exchange but, once installed, it occupies no hub memory except for one long used to communicate with the rest of the Propeller. It maintains a queue of up to 15 keystrokes in cog memory.

Programs that just want to use sxxb's functionality should use the sxxb interface object, [isxxb](#). Only programs that need to install the sxxb driver, such as sphinx.bin, should use the sxxb object itself.

Spin interface methods

- PUB **start**(*pingroup*) -- starts the keyboard driver. An even value for *pingroup* indicates a Demo/Protoboard-compatible keyboard interface; an odd value indicates a HYDRA-style keyboard interface.
- PUB **startx**(*pingroup*, *locks*, *auto*) -- similar to **start** but also lets you specify lock key behavior and auto-repeat timing. See sxxb.spn for details.
- PUB **peekkey** -- returns the next keycode but does not remove it from the queue. Returns 0 if no key pressed.
- PUB **key** -- returns the next keycode. Returns 0 if no key pressed.
- PUB **getkey** -- returns the next keycode. If no keys are in the queue, this method will wait until one is.

Implementation notes

The rendezvous location for sxxb is \$7ff8. If no keyboard input is available, the value at the rendezvous remains 0. If keyboard input is available, sxxb stores the next keycode at the rendezvous.

A program consuming keyboard input should wait for a non-0 value at the rendezvous, read the non-0 keycode, then clear the rendezvous to 0 so that sxxb can set it to the next keycode.

Sxxb is a moderately modified version of comboKeyboard by Mike Green.

sxtv

Sxtv is the Sphinx TV driver. It displays 13 lines of 40 characters. It is functionally very similar to the standard tv_text driver from Parallax, but it maintains its screen buffer in cog memory. Once installed, it occupies no hub memory except for a single long.

Programs that just want to use sxtv's functionality should use the sxtv interface object, [isxtv](#). Only programs that need to install the sxtv driver, such as sphinx.bin, should use the sxtv object itself.

Spin interface methods

- PUB **start**(*basepin*, *rv*) -- Sphinx.bin calls this to start the driver. The first parameter is the video base pin (this is hard-coded into sphinx.spin at [installation](#)). The second parameter is the rendezvous location, defined in Sphinx as \$7ffc (but in theory you could use sxtv in your own programs and use any long as the rendezvous).
- PUB **stop** -- stops the driver.
- PUB **GetBasepin** -- returns the video base pin (set by the call to **start**).
- PUB **str**(*stringptr*) -- prints the null-terminated string at address *stringptr*.
- PUB **dec**(*value*) -- prints *value* as a decimal number.
- PUB **hex**(*value*, *digits*) -- prints *value* as a hexadecimal number of *digits* digits.
- PUB **bin**(*value*, *digits*) -- prints *value* as a binary number of *digits* bits.
- PUB **out**(*c*) -- prints the character *c*. \$08 is interpreted as backspace, \$0d is interpreted as a carriage return.

Implementation notes

Once sxtv is installed by sphinx.bin, it monitors the long at hub address \$7ffc (known as the *rendezvous*, as in a meeting place; this is where sxtv and the rest of the Propeller meet to exchange information). As long as that long is 0, sxtv does nothing. Generally, sxtv waits for a non-0 value, does something with that value, then resets the long to 0 and resumes waiting.

Conversely, a program that wants to send information to sxtv waits for the rendezvous long to become 0, then sets it to a non-0 value.

If the long is a number between 1 and 255, sxtv prints the corresponding character (ASCII plus special Parallax characters) on the screen. Sxtv interprets backspace (\$08) and carriage return (\$0d) only.

If the long is "D"<<8, sxtv disables its video output. It still updates its internal screen buffer. This is useful for programs such as [ed](#) that use their own video drivers.

If the long is "E"<<8, sxtv enables its video output. This is the default state.

If the long is -1, sxtv sets the long to basepin<<8 and does not clear it. In this one situation, the long is used to convey information out from sxtv. Typically sphinx.bin uses this to determine whether sxtv is already running. Ed uses this to retrieve which pins the video hardware uses.

The preceding details are hidden by convenient PUB methods; however, note that communication with sxtv is not limited to Spin programs. Any cog can access the rendezvous location. This means that a PASM program can output to the screen (very handy for debugging).

Sxtv.spn is a heavily modified version of tv.spin by Chip Gracey.

Device interfaces

isxfs

Isxfs provides a convenient low-level interface to the Sphinx file system ([sxfs](#)). You must provide a 136-long area of memory (known in Sphinx as *filestuff*) for each file you want to open and pass the address of that area to each isxfs method you use. For a higher-level file object, see [sxfile](#).

Spin interface methods

- PUB **Open**(*pFilestuff*, *pFilename*, *mode*) -- opens a file. *mode* is either "R" or "W". Returns 0 on success. If opening for reading, returns 1 if file not found.
- PUB **Close**(*pFilestuff*) -- closes a file.
- PUB **Read**(*pFilestuff*, *pBuffer*, *nBytes*) -- reads *nBytes* bytes from a file into hub memory starting at *pBuffer*.
- PUB **Write**(*pFilestuff*, *pBuffer*, *nBytes*) -- writes *nBytes* bytes to a file from hub memory starting at *pBuffer*.
- PUB **Execute**(*pFilestuff*, *execmode*) -- loads a file into memory and executes it. If *execmode* is non-0, perform the equivalent of a system reset before starting execution.
- PUB **Command**(*cmd*, *param0*, *param1*, *param2*) -- execute an sxfs command.

isxkb

Isxkb is a lightweight interface object between Sphinx-aware programs and the Sphinx keyboard driver ([sxkb](#)).

Note that any object in a program's object hierarchy can use isxkb, not just the top object. This allows keyboard input anywhere in a program and can be very useful.

Spin interface methods

- PUB **peekkey** -- returns the next keycode but does not remove it from the queue. Returns 0 if no key pressed.
- PUB **key** -- returns the next keycode. Returns 0 if no key pressed.
- PUB **getkey** -- returns the next keycode. If no keys are in the queue, this method will wait until one is.

Example

```
' A simple demonstration of isxkb
' Note: Sphinx-aware programs don't need clock
settings
obj
  kb : "isxkb"
  term : "isxtv"
  f: "sxfile"
pub Main | ch
  term.str( string("Are you alive? ") )
  ch := kb.getkey
  term.out( ch ) ' echo keystroke
  term.out( 13 )
  if ch == "y" or ch == "Y"
    term.str( string("That's a relief", 13) )
  else
    term.str( string("Uh-oh", 13) )
  ' Our work here is done. Now back to the command
shell:
  f.Open( string("sphinx.bin"), "R" )
  f.Execute( 0 )
```

isxtv

Isxtv is a lightweight interface object between Sphinx-aware programs and the Sphinx TV driver ([sxtv](#)).

Note that any object in a program's object hierarchy can use isxtv, not just the top object. This allows output from anywhere in a program and can be very

useful.

Spin interface methods

- PUB **GetBasepin** -- returns the video base pin.
- PUB **str(stringptr)** -- prints the null-terminated string at address *stringptr*.
- PUB **dec(value)** -- prints *value* as a decimal number.
- PUB **hex(value, digits)** -- prints *value* as a hexadecimal number of *digits* digits.
- PUB **bin(value, digits)** -- prints *value* as a binary number of *digits* bits.
- PUB **out(c)** -- prints the character *c*. \$08 is interpreted as backspace, \$0d is interpreted as a carriage return.

Example

```
' A simple demonstration of isxtv
' Note: Sphinx-aware programs don't need clock
settings
obj
  term : "isxtv"
  f: "sxfile"
pub Main
  term.str( string("Howdy") )
  term.out( 13 )
  ' Our work here is done. Now back to the command
shell:
  f.Open( string("sphinx.bin"), "R" )
  f.Execute( 0 )
```

sxfile

Sxfile is a file object. Use it to read, write, or execute files on the SD card.

Each sxfile instance in a program refers to a different file, so you can have multiple files open concurrently.

Spin methods

- PUB **Open(pFilename, mode)** -- *pFilename* points to a null-terminated 8.3 filename (e.g., "test.spn"). *mode* is either "R" or "W". If *mode* is "R" and the specified file does not exist, this method returns 1; otherwise, this

- method returns 0 to indicate success and aborts if an error occurs.
- **PUB Close** -- closes an open file. You must close files that have been opened for writing in order to flush buffers to the SD card. Closing files opened for reading is less important (actually optional).
 - **PUB Read**(*ptr*, *n*) -- read *n* bytes into hub memory starting at address *ptr*. Returns the number of bytes actually read, or -1 if the end of file is reached.
 - **PUB Write**(*ptr*, *n*) -- write *n* bytes from hub memory starting at address *ptr*.
 - **PUB Length** -- returns the length of the file. The file must first be opened for reading.
 - **PUB Execute**(*mode*) -- executes the file. The file must first be opened for reading. A non-0 *mode* causes the equivalent of a system reset before the file runs.
 - **PUB ReadString**(*p*, *MAXLENGTH*) -- reads a null-terminated string into hub memory starting at address *p*. If the string exceeds *MAXLENGTH* characters, this method aborts.
 - **PUB ReadStringUpperCase**(*p*, *MAXLENGTH*) -- similar to **ReadString** but converts all lower-case characters to upper-case.
 - **PUB WriteString**(*p*) -- writes a null-terminated string to the file (including terminating null).
 - **PUB ReadNumber** -- reads a null-terminated string representing a decimal number and returns the numeric value (e.g. if the file contains "123" followed by a null byte, this method will return 123). This method aborts if a non-numeric character is encountered.
 - **PUB ReadByte** -- reads a byte from the file.
 - **PUB WriteByte**(*b*) -- writes a byte to the file.
 - **PUB ReadWord** -- reads a 2-byte quantity from the file.
 - **PUB WriteWord**(*w*) -- writes a 2-byte quantity to the file.
 - **PUB ReadLong** -- reads a 4-byte quantity from the file.
 - **PUB WriteLong**(*l*) -- writes a 4-byte quantity to the file.
 - **PUB SkipBytes**(*n*) -- skips over the next *n* bytes in the file.

PC utilities

Although Sphinx is ultimately intended to enable standalone Propeller development, it is undeniably convenient to be able to transfer files to and from the PC via USB. The two utility programs described here are run on the PC. Sphinx.bin must be running on the Propeller.

get

Transfers a file from Sphinx to the PC.

Usage:

```
get portname filename
```

portname is the serial port connected to the Propeller. *filename* is the name of the file on the SD card to transfer to the PC.

Example:

```
get com7 hello.spn
```

This command transfers hello.spn from the SD card to hello.spn on the PC.

Source code for get is provided on the [downloads](#) page. It is a C# program requiring .Net 2.0 or later. It runs on Windows and should run on Linux with Mono. Conversion to other languages should be straightforward.

put

Transfers a file from the PC to Sphinx.

Usage:

```
put portname filename [filename2]
```

portname is the serial port connected to the Propeller. *filename* is the name of the file on the PC to transfer to the SD card. If *filename* ends with ".spin", ".binary", or ".eeprom", the file on the SD card will have a name ending in ".spn", ".bin", or ".eep". Optionally, you can specify the filename you want the SD card file to have (*filename2*).

Example:

```
put com7 hello.spin
```

This commands transfers hello.spin from the PC to hello.spn on the SD card. Source code for put is provided on the [downloads](#) page. It is a C# program requiring .Net 2.0 or later. It runs on Windows and should run on Linux with Mono. Conversion to other languages should be straightforward.