Compiling and linking: a SOB story

The process of building a program with Sphinx is divided into two steps, compiling and linking. Compiling transforms a Spin source code into a Spin Object Binary (SOB) file. Linking converts a SOB and its sub-SOBs into an executable binary image.

Compiling

The Sphinx "compiler" proper actually consists of two programs, lex.bin and codegen.bin, but they work so closely together that they are best considered a unit. Lex.bin reads a .spn file, tokenizes it, and produces an intermediate .tok file. Codegen.bin reads the .tok file, parses it, and generates object code which it saves in a .sob file.

A SOB contains the Spin bytecodes for the object's PUB and PRI methods. In addition, it contains a list of the object's *imports* (sub-objects) and its *exports* (constants and PUB method signatures).

The compiler only compiles a single .spn file at a time. If the .spn file contains a sub-object, the compiler reads the .sob file for that sub-object and retrieves its exported information. This gives the compiler enough information to compile the .spn file without having to compile additional .spn files.

This also means that objects have to be compiled from the bottom up. That is, sub-objects have to be compiled before any containing objects are compiled.

Consider a hypothetical object, A, that has a sub-object, B. B in turn has a sub-object C. When you compile A.spn, the compiler will need to read B.sob, so you have to have compiled B.spn beforehand. Similarly, before compiling B.spn, you must compile C.spn to produce C.sob. So to compile A from a standing start you must issue these commands:

```
c C
c B
cl A
```

Of course, subsequently you will not have to compile all three objects every time, just the objects that change.

Linking

You invoke the linker (link.bin) with the name of the top-level object. The linker reads the SOB and goes through the SOB's list of imports (in other words, its sub-objects). The linker recursively reads the sub-SOBs and their sub-SOBs.

Once all the SOBs are read, the linker determines how it will lay out all the objects' bytecode in memory and adjusts the various inter-object pointers so that they all refer to one another correctly. Then it writes an executable binary image (.bin file).

Timestamps

Consider the following situation:

1. sub.spn is compiled, producing sub.sob.
2. top.spn is compiled using sub.sob, producing top.sob.
3. sub.spn is modified and recompiled, producing a new sub.sob.

Now the top SOB is based on an old version of the sub SOB. When the objects are linked, the resulting executable file may well fail because of that version mismatch.

In order to detect such version mismatches, Sphinx maintains a 32-bit "timestamp" in a file named timestmp.d8a. Every time you compile an object, Sphinx increments the timestamp and stores it in the .sob file. (Of course a real timestamp could be used, but Sphinx takes this approach so as not to require a real-time clock.)

The linker compares timestamps and warns if any object being linked is out of date with respect to a sub-object.

Note that this timestamp mechanism cannot detect when a .sob file is out of date with respect to its .spn counterpart. That is, Sphinx cannot tell if a .spn file has been modified after being compiled. If you edit a source file, it is your responsibility to remember to compile it.

Next: The Sphinx operating environment