SUBQMOD'S
LINKER
FOR THE
ATARI JAGUAR

Programmers Reference Manual
For Release 0.1.4 (Beta)

SubQMod's Linker for the Atari Jaguar

Introduction

This document describes SLN, **S**ubQMod's **L**i**N**ker (pronounced 'ess-ell-en'), which takes object modules created by an assembler or high-level language compiler and links them together to form a single executable program file for the classic and powerful Atari Jaguar console. SLN can also link in binary files created by art tools, music tools, sound tools and other such programs. By accepting these files directly SLN can save you time and disk space.

SLN is designed to be familiar with existing Jaguar developers by operating in a near identical manner to ALN, Atari's original Jaguar linker. In fact, the foundation to SLN is the ALN source code that existed prior to being modified for the Atari Jaguar and is used with the kind permission of its author. SLN is partially an homage to ALN and the fantastic Atari Jaguar console as well as a platform to provide Windows, MAC and Linux users with the capability of once more creating great code for this great system.

The Command Line

Below is the basic format of the SLN command line:

```
sln [options] <input files>
```

SLN understands a wide variety of command line switches which affect its mode of operation. These are listed and described in the next section.

SLN creates COFF encapsulated format executable files either with or without symbols and debugging information.

Command Line Switches

A summary of the available command line switches is shown below. Note that all of these options **must** be specified before any of the input files are listed, with the exceptions of the **–x**, **-l** and **–ii** options.

Switch	Description
-? or –h	Display SLN usage information. Use -? for Windows based systems and –h for all other systems.
-a *text, data, bss*	Output absolute executable file. `text` = Address for TEXT segment `data` = Address for DATA segment `bss` = Address for BSS segment Values for *text*, *data* and *bss* can be: a hexadecimal value to be used as the address **r:** relocatable segment (not useful for Jaguar programs) **x:** contiguous segment (contiguous with previous segment) For example "-a 802000 x 4000" would put the TEXT segment at $802000, the DATA segment immediately after that and the BSS section at $4000.
-b	Don't remove multiply defined local labels
-c*[fname]*	Add contents of the *fname* to the command line. They are read and processed as though they appeared on the command line. Any command line options may be used. Arguments in the file may be delimited by whitespace (tabs, spaces, newlines) or commas. As with the regular command line, only the **–i**, **-ii** or **–x** options may be used after the first input file is specified.
-d	Wait for keypress before exiting, after link is finished. This gives the user time to read any error messages. This can be useful if running SLN directly from a graphic user interface instead of a command prompt. Note that if you start SLN with no arguments (entering interactive mode), then the **–d** option is implied.
-e	Output COFF encapsulated executable (absolute only, must be used with the **–a** option).
-g	Output source-level debugging information (only works with the **–e** option to produce COFF format executable files).

-i *fname label* **-ii** *fname label*	Includes the binary data contained in the file specified by *fname* in the link. The contents of the file are placed verbatim into the DATA section. SLN creates a global symbol name *label* with the value of the starting address and another global symbol name *labelx* with the value of the ending address + 1 (e.g. if *label* is "picture" then you get a label named "***picture***" at the start and a second label named "***picturex***" at the end). With the **–i** option, the symbol created will be truncated to a maximum of 8 characters in length (the end symbol will be truncated to 7 before the 'x' for a total of 8 characters). With the **–ii** option, the symbol will not be truncated (assuming that you have specified COFF-format output). This option is used within the list of input files. Its similar to the SMAC/MADMAC directive **.incbin**.
-l	Add local symbols to output file as well as global symbols. This option is like a stronger version of the **–s** option.
-m	Produce load symbols map on standard output. The load map contains each symbol's name, value and type. The load map lists only global symbols unless the **–l** option is used.
-n	Output no file header to ABS file (output raw image of TEXT and DATA sections).
-o *fname*	Set the output filename to *fname*.

-r[*size*]	Section alignment size. Automatically pad the size of each object modules TEXT, DATA and BSS sections so that the size is an integral multiple of the specified size. Size is one of:
	
	`w: word (2 bytes)`
	`l: long (4 bytes)`
	`p: phrase (8 bytes, default alignment)`
	`d: double phrase (16 bytes)`
	`q: quad phrase (32 bytes)`
	
	For example, the option **–rp** would cause the TEXT, DATA and BSS sections of each object module in the link to be padded in size until they were a multiple of 8 bytes.

-s	Generate a symbol table in the output file and include all global symbols. Use the **–i** option by itself to include local symbols as well as globals.
-v	Set verbose mode. Repeating this option increases the information displayed on the standard output.
-z	Suppress information banner.

Using SLN

Below is a sample command line passed to SLN:

```
sln –e -l –rp –v –v –a 802000 x 4000 –o showimg.cof start.o
draw.o init.o video.o sound.o objlist.o –i image.dat img_data
```

This would run SLN with options for COFF output (**-e**), place symbols in the output file (**-l**), include local symbol (**-l**), align each segment of each file on a phrase boundary (**-rp**), show extra verbose information (**-v –v**), create an absolute executable file with TEXT and DATA segments starting at $802000 and a BSS segment at $4000 (**-a 802000 x 4000**) and output to SHOWIMG.COF (**-o showimg.cof**).

The input modules would be START.O, DRAW.O, INIT.O, VIDEO.O, SOUND.O and OBJLIST.O. Also included would be the binary data file IMAGE.DAT which would be reference via the *img_data* label.

The command lines can become unnecessarily long and tedious to type. To get around this, we need to have a linker command file that specifies some of the command line options and/or input files. Normally, you would specify your options in the first part of the command line and put the names of your input files into the linker command file. So we would probably really do something like this instead:

```
sln –e -l –rp –v –v –a 802000 x 4000 –o showimg.cof –c
showimg.lnk
```

The first part of the command line is the same, but then it ends with the **–c showimg.lnk** option instead of a list of input files to be linked. This option tells SLN that there are more linker commands in the text file SHOWIMG.LNK. This file would contain something like this:

```
start.o draw.o init.o
video.o sound.o objlist.o
–i image.dat img_data
```

The command file can be as long as required to specify all of your input files and options.

Filenames and the Library path

SLN looks for files in both the current default directory and in the directory named as the *library path*. This is specified by the **SLNPATH** environment variable.

The library path should be a full pathname which names a single directory, like "C:\JAGUAR\LIB". The complete path, including the drive letter, should be specified.

BSD/COFF File Formats

The primary style of file format supported by SLN is the BSD/COFF format.

Error Messages

Most of SLN's error messages are self-explanatory. They fall into four classes: warnings about situations that you (or the linker) may not be happy about, errors that cause the linker to not generate executables files, fatal errors that cause the assembler to abort immediately, and internal errors that should never happen.

You can write editor macros (or sed or awk scripts) to parse the error messages SMAC generates. When a message is printed, it is of the form:

```
filename[line number]: type: message
```

The first element, a filename, indicates the file that generated the error. The filename is followed by the line number in the file (enclosed in square brackets), then a colon followed by a space. The type of error message (Error, Warning, Fatal etc) is displayed followed by a colon and space and then finally the error message itself.