

A FILE FORMATS

The development tools support many file formats, in some cases several for each development tool. This appendix describes the formats of files that you prepare as input for the tools and points out features of files, which are produced by the tools. The three types of file formats that you can learn about in this appendix are as follows:

- [“Source Files” on page A-2](#)
- [“Build \(Processed\) Files” on page A-6](#)
- [“Debugger Files” on page A-11](#)

Most of the development tools use industry standard file formats. Sources that describe these formats appear in [“Format References” on page A-12](#).

Source Files

Source files are files that you prepare as input for the development tools. This section describes the following types of input file formats:

- “C/C++ Source Files (.C, .CPP, .CXX)” on page A-2
- “Assembly Source Files (.ASM)” on page A-3
- “Assembly Initialization Data Files (.DAT)” on page A-3
- “Preprocessor Header Files (.H)” on page A-4
- “Linker Description Files (.LDF)” on page A-5
- “Linker Command Line Files (.TXT)” on page A-5

C/C++ Source Files (.C, .CPP, .CXX)

These are text files containing C/C++ code, compiler directives, possibly a mixture of assembler code and directives, and (typically) preprocessor commands.

Several “dialects” of C code are supported: pure (portable) ANSI C, and ANSI C with ADI extensions. These extensions include memory type designations for certain data objects, and segment directives used by the linker to structure and place executable files.

For information on using the C and C++ compilers and associated tools, as well as definitions of ADI extensions to ANSI C, see the *C/C++ Compiler & Library Manual for TigerSHARC DSPs*.

For information on specifying the C dialect and general C/C++ code handling within VisualDSP++, see the *VisualDSP++ User’s Guide*.

Assembly Source Files (.ASM)

Assembly source files are text files containing assembly instructions, assembler directives, and (optionally) preprocessor commands. For information on assembly instructions, see the *Instruction Set Reference* manual for the corresponding DSP.

The DSP's instruction set is supplemented with assembler directives. Preprocessor commands control macro processing and conditional assembly. For more information on the assembler and preprocessor, see the *Assembler and Preprocessor Manual for TigerSHARC DSPs*.

Assembly Initialization Data Files (.DAT)

Data files are ASCII text files that contain fixed- or floating-point data. These files can provide the initialization data for an assembler `.VAR` directive or serve in other tool operations. When a `.VAR` directive uses a `.DAT` file for data initialization, the assembler reads the data file and initializes the buffer in the output object file (`.DOJ`). Data files have one data value per line and may have any number of lines.

Floating-point values in data files consist of a decimal mantissa value with a decimal point and a decimal integer exponent value. The exponent is separated from the mantissa with an upper or lower case “e.” The mantissa and exponent can be signed. The following are all valid floating-point values:

```
1.2345e12
567.8e-3
-7.1234
```

The assembler converts these decimal representations to floating-point format, either standard IEEE 32-bit single precision or 40-bit extended precision. The source code file in which a `.VAR` initialization occurs specifies the format (32-bit or 40-bit) for floating-point numbers using the

Source Files

`.PRECISION` directives. If a floating-point number in an initialization file does not fit in the specified format, the assembler rounds the number.

The rounding mode is also specified in the source code file with a `.ROUND` directive. For more information, see the *Assembler and Preprocessor Manual for TigerSHARC DSPs*.

The assembler supports binary, decimal, and hexadecimal numeric formats (bases) within expressions and assembly instructions. [Table A-1](#) describes the conventions of notation the assembler uses to distinguish between numeric formats.

Table A-1. Numeric Formats

Convention	Description
<code>0xnumber</code>	An “0x” prefix indicates a hexadecimal <i>number</i> .
<code>B#number</code> <code>b#number</code>	A “B#” or “b#” prefix indicates a binary <i>number</i> .
<code>number</code>	No prefix indicates a decimal <i>number</i> .
<code>numberr</code>	An “r” suffix indicates a fractional number.

Note that if any operand in an expression is a floating-point value, the assembler stores the value of the expression in floating-point format.

For all numeric bases, the assembler uses 32-bit words for data storage.

Preprocessor Header Files (.H)

Header files are ASCII text files that contain macros or other preprocessor commands that the preprocessor substitutes into source files. For information on macros or other preprocessor commands, see the *Assembler and Preprocessor Manual for TigerSHARC DSPs*.

Linker Description Files (.LDF)

The linker's .LDF files are ASCII text files that contain commands for the linker in the linker's scripting language. For information on this scripting language, see [“Linker Description File Reference” on page 2-38](#).

Linker Command Line Files (.TXT)

The linker's command line files are ASCII text files that contain command-line input for the linker. For more information on the linker's command line, see [“Linker Command-Line Reference” in Chapter 2 Linker](#).

Build (Processed) Files

Build files are files that the development tools produce when building your VisualDSP++ project. This section describes the following types of build file formats:

- [“Assembler Object Files \(.DOJ\)” on page A-6](#)
- [“Archiver Archive Files \(.DLB\)” on page A-6](#)
- [“Linker Executable Files \(.DXE, .SM, .OVL, .DLO\)” on page A-7](#)
- [“Linker Memory Map Files \(.MAP\)” on page A-7](#)
- [“Loader Hex Format Files \(.LDR\)” on page A-7](#)
- [“Loader ASCII Format Files \(.LDR\)” on page A-9](#)
- [“Loader Include Format Files \(.LDR\)” on page A-9](#)
- [“Loader Binary Format Files \(.LDR\)” on page A-10](#)

Assembler Object Files (.DOJ)

The assembler’s output object files are binary, Executable and Linkable File (ELF) format. Object files contain relocatable code and debugging information for your program’s segments. The linker processes your object files into an executable file. For information on the ELF and COFF formats that may be used for object files, see [“Format References” on page A-12](#).

Archiver Archive Files (.DLB)

The archiver’s output archive files are binary, Executable and Linkable File (ELF) format. Archive files contain one or more object files, called *Archive Elements*. The linker searches through archive files for any archive

elements that your code uses. For information on the ELF format that is used for executable files, see [“Format References” on page A-12](#).

Note that the archiver automatically converts input objects from COFF to ELF format.

Linker Executable Files (.DXE, .SM, .OVL, .DLO)

The linker’s output executable files are binary, Executable and Linkable File (ELF) format. Executable files contain your program’s code and debugging information. The linker may fully or partially resolve addresses in executable files depending on the options given on the linker’s command line and in your linker description file. For information on the ELF format that is used for executable files, see [“Format References” on page A-12](#).

Note that the linker automatically converts input objects from COFF to ELF format.

Linker Memory Map Files (.MAP)

The linker’s memory map files are ASCII text files that contain memory and symbol information for your executable file(s). The map contains a summary of memory that you define with `MEMORY{ }` commands in your linker description file, and provides a listing of the absolute addresses of all symbols.

Loader Hex Format Files (.LDR)

The loader’s output Hex-format files are in ASCII, Intel Hex-32 format. Hex files from the loader support 8-bit wide PROMs. The files are used with an industry-standard PROM programmer to program memory devices for your hardware system. One file contains data for the whole series of memory chips to be programmed.

Build (Processed) Files

The following example shows how the Intel Hex-32 format appears in the loader's output file. Each line in the Intel Hex-32 file contains either a data record, an extended linear address record or the end of file record:

:0200000040010E9	extended linear address record
:0A0004003C40343434261422260850	data record
:00000401FB	end of file record

The extended linear address record is used because a data record has only a 4-character (16-bit) address field, but the ADSP-TSxxx processors require 32 bits to address data memory and 24 bits to address program memory. The extended linear address record specifies address bits 16-31 for the data records that follow it. Data records are organized into the following fields:

:0A0004003C40343434261422260850	example record
:	start character
0A	byte count of this record
0004	address of first data byte
00	record type
3C	first data byte
08	last data byte
50	checksum

Extended linear address records have the following fields:

:02000000340850	
:	start character
02	byte count (always 02)
0000	address (always 0000)
00	record type
3408	offset address
50	checksum

The end of file record looks like this:

:00000001FF	
:	start character
00	byte count (zero for this record)
0000	address of first byte
01	record type
FF	checksum

Loader ASCII Format Files (.LDR)

The loader's ASCII-format file is similar to an assembler initialization data file (.DAT). The data order is one 16-bit hexadecimal value per line, providing lower-, middle-, then upper-16-bits of each 48-bit instruction. Use files of this format in the same manner as data files. For information on this format, see [“Assembly Initialization Data Files \(.DAT\)” on page A-3](#).

Loader Include Format Files (.LDR)

The loader's include-format file is an ASCII text file that consists of 48-bit instructions one per line with each instruction presented as three 16-bit hexadecimal numbers. The data order is lower-, middle-, then upper-16-bits of each 48-bit instruction. A few example lines from an Include format file appear as follows:

```
0x005c, 0x0002, 0x0620,
0x0045, 0x0000, 0x1103,
0x00c2, 0x0002, 0x06be
```

Build (Processed) Files

This file format lets you include the loader file in a C program. To include this file in a C program, use the following code:

```
#define My_Words = 2418
/* My_Words is the number of words in the loader file. Check
this number each time you generate a new loadable file. */

float my_loader_file [My_Words] = {
    #include "file.ldr"
};
```

Loader Binary Format Files (.LDR)

The loader's binary-format file supports a variety of PROM and micro-controller storage options and uses less space than the other loader file formats. The binary file contains 48-bit instructions in big-endian format (most significant bit first).

Debugger Files

Debugger files provide input to the debugger to define support for simulation or emulation of your program. The debugger supports all the executable file types produced by the linker (.DXXE, .SM, .OVL, .DLO). To simulate I/O, the debugger supports the data file formats (.DAT) from the assembler, the loadable file formats from the loader (.LDR), and the PROM formats from the splitter (.S_#, .H_#, .STK)

The standard hexadecimal format for a SPORT data file is a single integer value per line. Hex numbers do not require the 0x prefix to indicate hexadecimal. A value can have any number of digits but are read into the SPORT as follows:

- Hexadecimal number is converted to binary
- Number of binary bits read in matches the word size set for the SPORT, which starts reading from the LSB. The SPORT then fills with zeros values that are shorter than the word size or, conversely, truncates any bits beyond the word size on the MSB end.

For example, a SPORT is set for 20-bit words, and the data file contains hex numbers. The simulator converts these hex numbers to binary then fills/truncates to match the SPORT word size.

In the following file, the number A5A5 is filled and the number 123456 is truncated

Table A-2. SPORT Data File example

Hex Number	Binary Number	Truncated/Filled
A5A5A	1010 0101 1010 0101 1010	1010 0101 1010 0101 1010
FFFF1	1111 1111 1111 1111 0001	1111 1111 1111 1111 0001

Format References

Table A-2. SPORT Data File example

Hex Number	Binary Number	Truncated/Filled
A5A5	1010 0101 1010 0101	0000 1010 0101 1010 0101
5A5A5	0101 1010 0101 1010 0101	0101 1010 0101 1010 0101
11111	0001 0001 0001 0001 0001	0001 0001 0001 0001 0001
123456	0001 0010 0011 0100 0101 0110	0010 0011 0100 0101 0110

Format References

The following texts define industry standard file formats that are supported by VisualDSP++:

- Gircys, G. R. (1988) Understanding and Using COFF, O'Reilly & Associates, Newton, MA
- (1993) Executable and Linkable Format (ELF) V1.1 from the Portable Formats Specification V1.1, Tools Interface Standards Committee, available from <http://developer.intel.com/vtune/tis.htm>
- (1993) Debugging Information Format (DWARF) V1.1 from the Portable Formats Specification V1.1, UNIX International, Inc., available from <http://developer.intel.com/vtune/tis.htm>