SIGNAL™

Application Notes

Version 4.04 / 5.04
March, 2008

Engineering Design

Engineering Design
262 Grizzly Peak Blvd
Berkeley, CA 94708 USA
Tel/fax 510-524-4476
Email info@engdes.com
www.engdes.com

Table of Contents

Application Note 5

SIGNAL Sound File Format

Applies to: SIGNAL & RTS

Introduction

This note describes the SIGNAL binary sound file format and related concepts. See the SIGNAL and RTS user guides for background on importing and exporting sound and measurement data. Note also that SIGNAL can export and import sound data in Wave and AIFF file formats, which may avoid any involvement with the SIGNAL file header. Throughout this note, "SIGNAL" refers to the SIGNAL and RTS programs interchangeably, and "RTS" refers to both RTS and RTSD, unless otherwise noted.

Overview of SIGNAL File Types

Most commercial programs, including SIGNAL, write data in both binary- and text-file format. **Text files** are typically used to store measured parameters, sound similarity values, and other extracted parameters, as well as general text information. They are created and read by the Write ASCII (WA) and Read ASCII (RA) command families in SIGNAL and the log-file storage command (L) in the RTS. Text files are virtually computer-independent, and they are the common denominator of file exchange among commercial statistics, spreadsheet, database, and graphics programs. They can be transferred easily between PC and Macintosh environments. And they can be easily viewed by the user, via TYPE or PRINT commands in DOS, or altered using a text editor such as DOS EDIT.

Binary files are normally used for storing and exchanging sampled sound signals. They are created and read with the Write (W) and Read (R) commands in both SIGNAL and the RTS. They are much more space-efficient than text files. Binary files are typically written in one of two forms - with a proprietary file header containing information such as size, sample rate, and binary coding, or headerless, i.e., with no header at all. SIGNAL can read and write binary sound files in headerless format or with SIGNAL, Wave, and AIFF headers. The RTS can read sound files with SIGNAL, Wave, or BLB headers, and can write sound files either headerless or with SIGNAL headers. **Header-based files** may require programming to enable exchange between systems, while **headerless files** are a default format for exchanging sampled sound data. However, file information such as sample rate and binary coding must be provided to the file-reader in order to interpret the headerless data. The SIGNAL header structure is the subject of this note.

Sound File Structure

SIGNAL sound files consist of two parts, a header section and a data section. The header section contains information about the file and its contents: file size, data representation (integer vs. real), buffer type (T, F, FT), sample rate, etc. The header mixes ASCII and floating point (real) data, and can be extended by the user to add demographic information or notes about the data. The data section contains the sampled waveform, power spectrum, or spectrogram, and is written in binary format, without record marks or other dividers. Data may be represented in either integer or floating-point (real) format. RTS sound files are identical, except that they only contain integer, time data.

Terminology

The following terms are used in this note:

• 1 File Block	= 512 bytes in length
and can contain:	128 4-byte (32-bit) real numbers 256 2-byte (16-bit) integers 512 1-byte ASCII characters
• 1 Header Element	= 4 bytes
and can contain:	1 real number 4 ASCII characters

• Abbreviations:

elt	= element
< >	= blank
N	= numerical header element = 4-byte real number
I	= numerical header element = 4-byte integer number
L	= literal header element = 4 ASCII characters

Header Section

The header is a mixture of 4-byte numerical (N) and literal (L) elements. Numerical elements contain one real number in 32-bit floating-point format, and literal elements contain 4 or fewer ASCII characters. Fewer than 4 ASCII characters are left justified with trailing spaces. Elements marked with an asterisk (*) are mandatory in the header. Optional (non-*) elements are filled with spaces (decimal 32) if type L, or zeroes if type N. Reserved elements are filled with zeroes.

The header specifies the lengths of the header and data sections in blocks (NHBLKS and NDBLKS), where 1 block = 512 bytes. The header section must contain an integral number of blocks. Through version 2.2, the SIGNAL data section contains an integral number of blocks, by padding the last block with zeroes. In SIGNAL versions 3.0 and later, and in the RTS, the final data block is not padded.

The default header is two blocks long, but the user may extend the header with additional blocks of information (such as demographic data or notes) by adjusting NHBLKS accordingly. SIGNAL will read the first two header blocks, skip (NHBLKS-2) blocks, then read NDBLKS blocks of data.

The first section in the header contains file attributes. See the detailed outline below. **PGM_STAMP** specifies which program created the file, and **PGM_VERSION** specifies the program (and file) version. **NHBLKS** and **NDBLKS** specify the number of header and data blocks in the file (see above). **BUFFER_TYPE** specifies the buffer type (T, F, or FT) and **DATA_TYPE** specifies the type of the data (integer or real).

For integer time files, **CNVFAC** and **OFFSET** specify the conversion factor and offset of the data, used to convert the integer data values to voltages. CNVFAC and OFFSET depend on the A/D converter. CNVFAC expresses Volts/bit, equal to the converter's input in Volts divided by its output in bits, for example, 10 Volts / 2048 = .00488 Volts/bit. OFFSET is the offset of the data from binary 0. For example, 12-bit data may be coded between either -2048 and +2047 or 0 and 4095. OFFSET expresses the midpoint of the coding range, e.g., 2048 for 0 to 4095 data. These factors are applied when the data is read into a SIGNAL buffer:

Buffer value (Volts) = (File value - OFFSET) * CNVFAC

Beginning with SIGNAL 4.02, SIGNAL sound files may contain multi-channel data. Data channels are interleaved – point 1 of channel 1, point 1 of channel 2, …, point 1 of channel N, point 2 of channel 1, etc. **NCHAN** specifies the number of data channels in the file.

The next section in the header contains buffer attributes. Again, see the detailed outline below. SIGNAL reads these directly into the Buffer Directory, so that the BD is completely recovered when a file is read in. For a complete description of these attributes, see the **SIGNAL User Guide**. Mandatory header elements for all buffer types (T, F, and FT) are **TPNTS**, the number of data points **per channel** (**not** the total number of data points in the file), **SRATE**, the sample rate of the data in points/sec, and **XLOW** and **XRANGE**, the time origin and time range of the data in msec. FT buffers also require **NTIME** and **NFREQ**, the number of time and frequency cells in the matrix, **XFTLEN**, the FFT length of the spectrogram, and **YLOW** and **YRANGE**, the frequency origin and frequency range of the data.

Data Section

The data section is written as continuous numerical data, with **no intervening record or block marks**, in either 16-bit integer or 32-bit floating point format (depending on **DATA_TYPE** in the header). Floating-point numbers are represented in IEEE 754 format. See Application Note 11, "Exchanging Data Files between SIGNAL and MATLAB", for a detailed discussion of floating-point representation.

Header Format

Following is the format of the two header blocks in the SIGNAL sound file. Again, note that elements marked with an asterisk (*) are mandatory. Unused elements are filled with spaces (hex 20) if literal (type L), or zeroes if numerical (type N). Literal elements are padded with trailing blanks. Reserved elements are filled with zeroes.

File Header Block #1

File Attributes

Elt No.	Type	Title	Contents	
1 *	L	PGM_STAMP	SIG	(SIGNAL)
			RTS	(RTS or RTSD)
			EXT	(External)
2	L	PGM_VERSION	x.xx	(Version)
3 *	N	NHBLKS	No. header blocks	
4 *	N	NDBLKS	No. data blocks	
5 *	L	BUFFER_TYPE	T = Time	
			F = Frequency	
			FT = Frequency-Time	
6 *	L	DATA_TYPE	I = Integer	
			R = Real	
7 *	N	CONV_FACTOR	[Integer time file only]	
8 *	N	OFFSET	[Integer time file only]	
9 *	N	NCHAN	[Time file only]	
10-20	N	Reserved		

Buffer Attributes

All Buffer Types

21 *	N	TPNTS	No. data points (per channel)
22 *	N	SRATE	Sample rate (pts/sec)
23 *	N	XLOW	X-origin (msec)
24 *	N	XRANGE	X-range (msec)
25-26	L	QTY	Physical quantity (e.g. AMPL)
27-28	L	UNITS	Physical units (e.g. VOLTS)
29	L	SCALE	< > = linear DB = log
30-34	L	TITLE	Buffer title

35	L	MODE	< > = Not complex
			R = Real part, complex
			I = Imag part, complex
			M = Magnitude, complex
			P = Phase, complex
			N = Normalized mag, complex

36	L	WINDOW	RECT = Rectangular
			HANN = Hanning
			HAMM = Hamming
			[F & FT only]

37	N	SMOOTHING	Smoothing width (msec,Hz)
38-39	L	{ DATE { Reserved	[3.0 and earlier] mm-dd-yy [3.1 and later]

40	L	XFSCALE	Transform energy scaling:
			PS = power spectrum
			PSL = PS w/ length scaling
			PSD = power spectral density
			PSDL = PSD w/ length scaling
			ES = energy spectrum
			ESD = energy spectral density

41-43	L	DATE2000	mm-dd-yyyy
44	I	TPNTS	No. data points (4-byte integer)
45	N	Reserved	

Temporary

[Defined in buffer directory, but not written into file header]

46	N	SBUF	Transform source buf no.
47-50	N	Reserved	

Buffer-Type Specific

Time & Frequency Buffers

51	N	ADBITS	A/D resolution [Integer time file only]
52-65	N	Reserved	

Frequency-Time Buffers

51 *	N	NTIME	No. time cells
52 *	N	NFREQ	No. frequency cells
53 *	N	XFTLEN	FFT length
54 *	N	YLOW	Y-origin (Hz)
55 *	N	YRANGE	Y-range (Hz)
56-65	N	Reserved	

Buffer Caption

66-85	L	CAPTION	80 characters

User-Defined Numerical Fields

86	N	BVAL1	1 real number
...		...	
105	N	BVAL20	1 real number

Reserved for System

106-128	N	Reserved	

File Header Block #2

User-Defined Label Fields

1-4	L	BLAB1	16 characters
...		...	
37-40	L	BLAB10	16 characters

User-Defined Axis Labels

41-55	L	XLAB	X-axis label (60 characters)
56-70	L	YLAB	Y-axis label (60 characters)

Reserved for System

71-128	N	Reserved

Format Differences between SIGNAL and RTS

SIGNAL and RTS sound files follow the same format, and can generally be interchanged freely. Note that the RTS reads and writes only time files, while SIGNAL writes time, spectrum, and spectrogram files. SIGNAL and RTS sound files differ in the following respects.

1. RTS sound files are written in integer format, while SIGNAL sound files may be stored in integer or real format. Thus SIGNAL will accept all RTS files, but to export from SIGNAL to the RTS, SIGNAL must store the file in integer format (using the INTWRT switch or W /I).

2. SIGNAL and the RTS write different file ID's in the PGM_STAMP and PGM_VERSION elements, which can be used to identify file origin. SIGNAL 3.1 writes "SIGP3.10" and RTSD 1.1 writes "RTS 2.20". SIGNAL 4.0 and RTS 4.0 write "SIGP4.0x" and "RTS 4.0x", where "x" is the minor version number.

Exchanging Sound Files with External Systems

Binary sound files can be exchanged between SIGNAL and other data acquisition systems. This requires that the sound files be in a format compatible with both systems. Approaches include: (1) write SIGNAL files in headerless binary format, (2) write SIGNAL files in Wave or AIFF format, (3) convert files from the SIGNAL format to the other system's format using a file converter, and (4) convert the other system's files to the SIGNAL format using a file converter. See the file import/export sections of the SIGNAL and RTS user guides for more detail. SIGNAL can read and write headerless binary files, and the RTS can write but not read headerless binary files. When writing a program to convert an external data file to the SIGNAL file format, keep the following points in mind.

1. Place the label "EXT" in PGM_STAMP and leave PGM_VERSION blank.
2. Determine CNVFAC and OFFSET for the external data, either empirically, or based on the known resolution and range of the external A/D converter.
3. Block and record marks may not appear anywhere in the file.

Programming Access to SIGNAL Files

The data section in the SIGNAL sound file is written as continuous numerical data, with no intervening record or block marks. Data can be read and written using any of the following software tools.

Software Environment	**Routines/Settings**
FORTRAN direct I/O	ACCESS=DIRECT, FORM=UNFORMATTED
C low-level I/O	open( ), read( ), write( )
DOS file services	Open file, read file, write file = DOS int 21h, functions 3ch, 3dh, 3eh, 3fh, 40h, and 42h

4-30-92
Rev 9-25-95
Rev 5-9-99
Rev 5-25-02
Rev 1-15-04
Rev 3-31-08 Caption extended from 72 to 80 characters

Application Note 6

Base Address, IRQ, and DMA Channel Usage

Applies to: SIGNAL & RTS

NOTE: This note applies only to DT-2821 and DART PCMCIA analog I/O boards.

Overview

Add-in hardware boards, such as analog I/O, DSP, SCSI controller, etc. are internally set to utilize a particular base address, interrupt level (IRQ), DMA channel, and/or ROM address. **Base address** sets the address through which the computer communicates with control registers on the board. Typical base addresses lie between 200 and 3f0 hex, and most boards occupy several addresses following the base, e.g., 240-24f hex (see the following table). **Interrupt level (IRQ)** indicates which interrupt channel will be used by the board to request interrupt service from the computer. IRQ channels are numbered 1 - 15 (decimal). **DMA channel** indicates which DMA (Direct Memory Access) channel will be used by the board to request DMA service. DMA channels are numbered 0 - 3 and 5 - 7. **ROM address** applies only to bootable controller boards, and indicates where the board's BIOS code is installed in the computer's memory. ROM addresses typically run in range c0000 - dffff (hex).

Hardware conflicts can arise when two different boards are set to the same base address, IRQ, DMA channel, or ROM address. These conflicts can cause the system to malfunction or even not boot. **Boards should never share a base address, IRQ, DMA channel, or ROM address setting**. Note that since a base address is really an address range (see the table), no part of that range may overlap another board's. Conflicts are resolved by determining the address/IRQ/DMA assignments of all boards in the system and altering these assignments to eliminate the overlap. This may involve changing both the board, through jumpers or software, and its controlling software, such as SIGNAL.CFG for the SIGNAL analog I/O board.

Base Address, IRQ, and DMA Assignments

The following table shows the base address, IRQ, and DMA assignments for the SIGNAL analog I/O and DSP boards and other common system components. If an IRQ or DMA entry is omitted in the table, then that board does not have IRQ or DMA hardware. **Note** that commercial programs for detecting address, IRQ, and DMA assignments are not reliable. The best way to resolve a conflict is to manually construct a similar table for your system, including all the system components. Note that post-1995 hard disk/CD-ROM drives may conflict with DT-2821 analog I/O boards.

Device	**Base Address Range**	**IRQ**	**DMA Channel**
Analog I/O:			
DT-2821 family	240 - 24f	15	5, 6
DART PCMCIA card	260 - 26f	10	
Arithmetic accelerator (for RTSD)	300 - 30f		
Sound chip on notebooks		5	
SCSI controller	340 - 35f	11	
CD-ROM, pre-1995		11	5
Hard disk / CD-ROM, post-1995		15	
System IRQ Assignments:			
Timer		0	
Keyboard		1	
IRQ Cascade		2	
COM2 (FAX/modem)	2f8 - 2ff	3	
COM1	3f8 - 3ff	4	
LPT2 (optional)		5	
LPT1	378 - 37b	7	
Floppy Disk		6	
Clock		8	
IRQ2 Redirect		9	
PCI bus interface		11	
Math Coprocessor		13	
Hard Disk, CD-ROM		14, 15	

5-9-94
Rev 9-25-95
Rev 4-18-99

Application Note 8

Analog I/O Board Diagnostics

Applies to: SIGNAL & RTS

NOTE: This note applies to DT-2821 analog I/O boards.

Introduction

If your DT-2821 analog I/O board appears to be malfunctioning, first check your signal source and all cables and instruments leading to or from the I/O board, including the SIGNAL I/O cable and panel. Remove any non-essential components and if possible exchange the others one by one. If you still suspect the board itself, then run the Data Translation diagnostics on the board. Following are instructions:

1. The DT diagnostics program "DT2821.EXE" is included in the C:\SIGNAL directory. To execute it, leave SIGNAL, attach to this directory, and type "DT2821" at the DOS prompt.

2. Select your board model from the menu: DT-21EZ (50 KHz), DT-2821 (50 KHz), DT-2821F (150 KHz), or DT-2821G (250 KHz). When asked if the board is factory configured, say YES unless you've rejumpered the board for a different base address or IRQ.

3. Run the following tests from the main menu and submenus. These tests require no external connections. Tests not listed below require that external signals be provided to the board. Within the submenus, it may be necessary to hit <arrow> keys and <enter> twice. After reporting test results (e.g., "Board Response PASSED") the program will pause before returning to the submenu - **don't hit <enter>**, just wait. **Note:** the DIO section of the Acceptance Test and the DIO test will fail - this is

normal, because the test requires special cabling. All other tests should pass. DIO is not involved in SIGNAL analog I/O operation.

Main Menu Test	**Submenu Test**
Acceptance Test	
Register Tests	Board Response
	ADCSR
	CHANCSR
	DA/DIOSR
	TMRCTR
	SUPCSR
Interrupt Tests	A/D Done Bit
	A/D Error Bit
	D/A Ready Bit
	D/A Error Bit
D/A Function Tests	DAC Ready Bit
	DAC Error Bit
A/D Function Tests	A/D Done Bit
	A/D Error Bit

If the diagnostics report a failure, US users should contact Data Translation for board repair (508-481-3700), and foreign users should contact their SIGNAL supplier - either Engineering Design or Noldus Information Technology. **Foreign users should not contact Data Translation, who is not responsible for the board.** If the board passes the DT diagnostics but errors persist, contact Engineering Design.

9-13-95

Application Note 10

Digital Transfer between SIGNAL and DAT Recorders

Applies to: SIGNAL & RTS

NOTE: This note has not been revised for SIGNAL 4.0.

Introduction

DAT (Digital Audio Tape) recorders store audio signals in digitized format, as a sequence of sampled data points, exactly like a SIGNAL or RTS time buffer. With additional hardware and software, this digitized sound data can transferred directly from the DAT recorder onto the user's hard disk without redigitizing. The resulting hard disk files can then be read into SIGNAL or the RTS for examination and analysis.

The principal advantage of direct digital transfer is sound accuracy. It avoids the potential degradation involved in converting the digital recording back to analog and then redigitizing in SIGNAL. Most important, it retains the 16-bit digitization of the DAT rather than the 12-bit coding of the standard SIGNAL analog I/O board. However, there are also several disadvantages. One is the extra cost of interface hardware and software. Another is that the signal's sample rate is fixed by the DAT recorder, which can waste buffer space, file size, and computation time. And while performing a direct digital transfer appears simpler than redigitizing, in fact, the multiple steps of starting Windows, performing the transfer, and possibly altering or decimating the sample rate can take more time than simply redigitizing the source material in SIGNAL from a DAT analog playback.

This note describes the hardware and software required to transfer digital sound material between a DAT recorder and SIGNAL, the steps involved in performing the transfer, and the performance of representative analog and DAT recorders and transfer components. Throughout this note, **"RTS" refers to both the RTS and RTSD software programs.**

Required Hardware and Software

Digital sound data is commonly transferred into and out of a DAT recorder via one of several interface standards. AES/EBU is the standard for professional audio, and normally uses 3-pin XLR connectors. IEC 958 is also known as S/PDIF (SONY/Philips Digital Interchange Format), and is the consumer standard. S/PDIF normally uses an electrical connection involving RCA phono connectors, but may also use an optical connection.

Digital transfer between a DAT recorder and computer involves connecting the recorder to the computer through a S/PDIF interface board, and requires the following components:

- DAT recorder with a S/PDIF interface
- S/PDIF digital interface PC-board
- Software for handling the digital transfer
- [optional] Sound card for listening to the transferred sound files
- [optional] Software for altering sound file sample rate

DAT Recorders

Many DAT recorders include a S/PDIF interface. Examples include professional machines such as the Panasonic SV-3800 and SONY PCM-R500, and portables such as the SONY TCD-D10. The S/PDIF interface is sometimes referred to as "COAXIAL" in DAT recorder product descriptions.

S/PDIF Interface Boards

The S/PDIF digital interface board (referred to here as the DIO board) converts digital data between a serial bit stream in the S/PDIF format and a sequence of digital words on the computer bus. One S/PDIF board that is widely used is the "**Card-D/Digital only**", model DO-01, manufactured by Digital Audio Labs, Plymouth, MN (612-559-9098, www.digitalaudio.com). The Card-D is an ISA board, and costs about $350. It can handle either of the two standard DAT sample rates, 44.1 KHz or 48 KHz. **Note:** Apparently when used with one of the more recent OEM versions of Windows 95/98 (the "OSR2" release), the Card-D requires an **older** driver (V1.30) than the one shipped with the board (either V1.31 or V1.40), which is available from the manufacturer's website.

Transfer Software

Transfer software controls data flow between the DIO board and computer memory. The digital transfer is performed under Windows, and the DIO board manufacturer normally includes a driver which makes the DIO board look like a conventional Windows sound card. Sound data is then transferred from the computer to the DAT by "playing" the sound file through the DIO board to the DAT, and sound data is transferred from the DAT to the computer by "recording" a sound file from the DAT through the DIO board. "Playing" and "recording" through the DIO board can be handled by the Windows Sound Recorder program, which is included with Windows. However, this program limits maximum sound duration per physical memory, and any other program that supports record and play functions through the Windows Sound System (such as Sound Forge, described below) should also work. These programs normally operate on Wave files, the Windows standard. See below for operating details.

Sound Card

A sound card can be useful to listen to the sound files received from the DAT, or check them before transfer to the DAT. One card that is widely used is the **Daytona PCI** sound card from Turtle Beach, Yonkers, NY (800-233-9377, www.tbeach.com). This is a PCI board, which is useful on newer computers with few ISA slots, and retails for about $125. If installing this card on a computer containing the SIGNAL analog I/O board, beware of potential hardware conflicts. See Application Note 6, "Base Address, IRQ, and DMA Channel Usage", for assistance. Windows can be set to play to either the sound card or DIO board via the Windows multimedia control panel.

Sample Rate Converter

Files transferred from a DAT recorder will be received in the native sample rate of the recorder, normally 44.1 KHz or 48 KHz. SIGNAL I/O boards provide sample rates resulting from integer division of 4,000,000 (e.g., 43956 = 4,000,000 / 91). Therefore the SIGNAL board cannot play DAT signals at their exact native sample rates. **Note** that this has no affect on the accuracy of SIGNAL's analysis capabilities. The following table gives the nearest available sample rate and the resulting percent error:

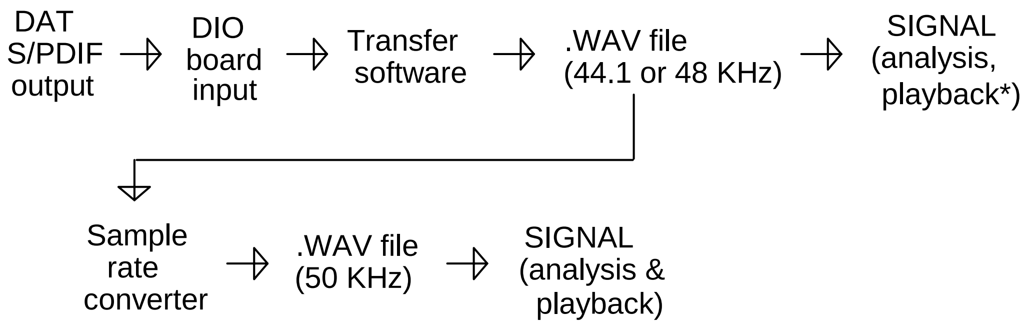
Native DAT sample rate	**Nearest SIGNAL sample rate**	**Percent error**
44100 KHz	43956	-.33 %
48000 KHz	48193	+.40 %

If the percent error is significant for your application, SIGNAL 4.0 can mathematically convert a digital signal from one sample rate to another. The process is based on mathematical function approximation and interpolation, and can be quite accurate. Inaccuracies introduce distortion in the resampled waveform, and resampling accuracy can be measured by the increase in total harmonic distortion (THD) between the original and resampled signals.

Operating Instructions

Transferring from DAT to SIGNAL

Following is a connection diagram for transferring sound material from the DAT recorder to SIGNAL. Sample rates are shown in ()'s.



*** = Playback at nearest available sample rate**

Running under Windows, the digital data flows from the S/PDIF output on the DAT recorder to the DIO board under control of transfer software such as Windows Sound Recorder or equivalent, into a Wave file on the computer disk. Following are suggested operating steps:

1. Connect the S/PDIF output on the DAT recorder to the DIO board input.

2. Start Windows and launch Sound Recorder. Make sure that Sound Recorder can find the DIO board and if necessary, set the Windows multimedia control panel to record from the DIO board.

3. Start recording in Sound Recorder, then start playback from the DAT. Stop both after the sound completes.

4. Save the recorded sound from Sound Recorder as a Wave file on disk.

5. Read the resulting Wave file into SIGNAL (using "R /W") or into the RTS (which will detect the file type automatically) for viewing, analysis, or measurement.

Using the DAT File in SIGNAL

The Wave file received from the DAT is limited in length only by the recording program and the available disk space. However, SIGNAL time buffers are limited to about 1.2 million data points, or about 27 seconds at 44.1 KHz sample rate. Longer files can be read into SIGNAL in segments using R /W /Q.

The Wave file will have the native sample rate of the DAT recorder, either 44.1 KHz or 48 KHz. Files at these sample rates can be viewed, measured, manipulated, modeled, edited, etc with full accuracy in SIGNAL, but playbacks will be performed at the nearest sample rate available on the SIGNAL I/O board, as indicated by the * in the connection diagram and discussed earlier. If this sample rate error is unacceptable, for example when making playback tapes, the Wave file can be resampled to an available sample rate such as 50 KHz using an external software sample rate converter. Both cases are shown in the connection diagram. See "Sample Rate Converter" for details.

Note that the sample rate of direct-from-DAT sound files may considerably exceed the bandwidth requirements of the recorded material. This can waste SIGNAL buffer space, disk space for storage, and computation time during analysis. In this case, sample rate can be decimated (reduced) by an integer multiple without loss of information, provided the signal is digitally filtered first. See "Time and Frequency Decimation" in Chapter 28, "Signal Processing", in the SIGNAL User Guide for details. Alternately, the DAT tape can be played into SIGNAL and redigitized at an appropriate lower sample rate.

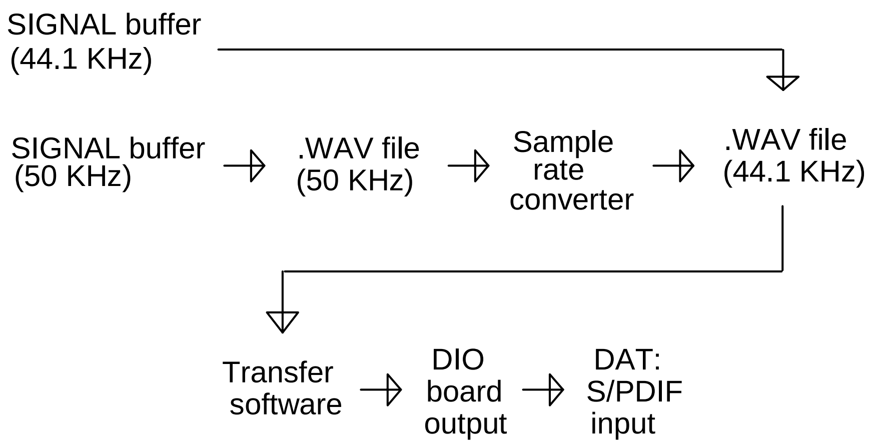
Using the DAT File in the RTS

The RTS can handle files up to 1 billion data points in length. It will detect the Wave file type automatically and read it the same as a SIGNAL file.

As in SIGNAL, Wave files in the native sample rate of the DAT can be viewed and measured in the RTS with full accuracy, but playbacks will be performed at the nearest available sample rate, as indicated by the * in the connection diagram. This should suffice for most purposes in the RTS. If exact playbacks are required, the Wave file can be resampled to a sample rate such as 50 KHz, as described in "Sample Rate Converter". Note that the RTS cannot write Wave files.

Transferring from SIGNAL to DAT

Following is a connection diagram for transferring sound material from SIGNAL to the DAT recorder. Sample rates are shown in ()'s.



The sounds of interest are stored from SIGNAL buffers as Wave files. Then, running under Windows, digital data flows from the Wave file on the computer disk to the DIO board, under control of transfer software such as Windows Sound Recorder or equivalent, to the S/PDIF input on the DAT recorder. Note that the RTS cannot currently write Wave files; however a file converter could be written to convert SIGNAL files from the RTS into Wave files.

Following are suggested operating steps:

1. Create a SIGNAL buffer containing the signal of interest and write it to a Wave file on disk using "W /W" in SIGNAL.

2. Start Windows and launch Sound Recorder. Make sure that Sound Recorder can find the DIO board and if necessary, set the Windows multimedia control panel to play through the DIO board. Queue up the desired Wave file for playback.

3. Connect the DIO board output to the S/PDIF input on the DAT recorder.

4. Start recording on the DAT, then start playback in Sound Recorder. Stop the DAT recorder after the sound completes.

The SIGNAL source buffer must either be created in the native sample rate of the DAT recorder (44.1 or 48 KHz), or else written to disk and then converted to this sample rate (see "Sample Rate Converter" above). Note again that SIGNAL buffer capacity is limited to about 1.2 million data points, or about 27 seconds at 44.1 KHz sample rate.

Performance Specifications

DAT Recorders

One of the attractions of DAT recorders is their superior performance relative to analog recorders. For interest, following are performance specifications for a few DAT recorders and a studio-quality Studer analog recorder. THD = total harmonic distortion. Values are from manufacturer's specifications except where noted as measured by "(meas)".

	SONY TCD-D10	**SONY PCM-R500**	**Panasonic SV-3800**	**Studer A810 @ 15 ips**
Record mode	Digital	Digital	Digital	Analog
Recorder type	Portable	Studio	Studio	Studio
Freq response	20-22 KHz, ±1 db	20-20 KHz, ±0.5 db	10-20 KHz, ±0.5 db	30-18 KHz, ± 1 db
Signal-to-noise	> 85 db	> 90 db	> 92 db	64 db (meas)
THD	-64 db (spec) -81 db (meas)	-66 db	-70 db at +4 dBu -83 db at +22 dBu	-57 db (meas)

The SONY TCD-D10 portable DAT recorder (cost about $800-1000) has become popular for field use, and its performance was characterized. Maximum input voltage = ±3.15 Volts, beyond which input signals are clipped. Measured THD is better than -81 db.

S/PDIF Board

The Card-D S/PDIF board was tested at the Borror Laboratory of Bioacoustics (BLB) at The Ohio State University. A Wave file containing a sine signal was created in SIGNAL, transferred digitally to the DAT, then transferred digitally back from the DAT to SIGNAL and analyzed for distortion. As expected, there was no detectable degradation in the returned signal.

8-30-98

Application Note 11

Exchanging Data Files between SIGNAL and MATLAB

Applies to: SIGNAL & RTS

Introduction

The MATLAB^{tm} program from The MathWorks (Natick, MA) is a programmable mathematical analysis and display language which provides a broad range of capabilities for mathematical modeling and signal analysis in a language environment similar to SIGNAL's. For the user willing to program, MATALB can add powerful capabilities to the SIGNAL environment – for example, extensive matrix operations (the "mat" in MATLAB), as well as application "toolboxes" covering neural networks, human speech, digital filters, a variety of spectral transforms, and other advanced signal analysis. SIGNAL users have employed MATLAB to perform linear predictive speech analysis, design and apply custom digital filtering, perform wavelet analyses of time signals, and perform matrix-based analysis of scanned physiology images.

SIGNAL users can access MATLAB's capabilities by exporting SIGNAL data files to MATLAB, performing desired analyses and signal transformations (such as filtering), then if necessary exporting the data back to SIGNAL. This is achievable because both programs provide basic and compatible facilities for importing and exporting data files. This application note describes how to exchange data between the two systems. The first part describes SIGNAL and MATLAB commands for data import and export, and the second provides background on data storage issues that affect this process.

Transferring Data between SIGNAL and MATLAB

Following is a discussion of the SIGNAL and MATLAB commands used to exchange data files between the two programs. Note that SIGNAL files can be stored in two data formats - **integer** or **floating point** - and two header formats - **SIGNAL file header** or **headerless**. See the chapter "File Storage and Exchange" in the **SIGNAL User Guide** for background. The basic strategies for file exchange are: 1) export data from SIGNAL in SIGNAL-header files and let MATLAB skip over the SIGNAL header and 2) export data from MATLAB in headerless files, supplying the essential file parameters when importing into SIGNAL.

Exporting integer SIGNAL time files to MATLAB

SIGNAL time buffers which have not been manipulated after acquisition can be transferred without loss of precision by using integer format, which stores the exact values delivered by the A/D converter.

SIGNAL-header integer files

SIGNAL-header integer files are read in MATLAB by skipping the header and applying a data conversion factor.

1. Write the SIGNAL time buffer to a SIGNAL-header integer time file using the /I flag:

```
>W T 1 /I
Filename: IFILE
```

2. Read the file into a MATLAB array. The following MATLAB commands will read the entire integer file IFILE into the array A. FSEEK skips the 1024-byte SIGNAL file header.

```
fid = fopen ('ifile','rb');
fseek(fid,1024,'bof');
A = fread (fid,inf,'int16');
```

3. SIGNAL-header integer files are stored in the native coding of the A/D converter (unlike headerless integer files - see below). 12-bit SIGNAL A/D boards produce data with an offset of 2048 and a gain of 2048 bits/10 Volts, while 16-bit SIGNAL A/D boards deliver an offset of 0 and a gain of 32768 bits/10 Volts. See "Import / Export of Binary Files" in the **SIGNAL User Guide**. These conversion factors are used to convert the file data to Volts after importing into MATLAB:

```
A = (A - 2048.) / 204.8;          12-bit A/D
A = A / 3276.8;                   16-bit A/D
```

4. **Example:** read and plot the sample sound file C:\SIGNAL\TWEET.1. To convert and display this file correctly, you must know its conversion format (12-bit A/D) and sample rate (25000 Hz).

```
fid = fopen ('c:\signal\tweet.1','rb');
fseek(fid,1024,'bof');
A = fread (fid,inf,'int16');
A = (A - 2048.) / 204.8;              % apply conversion factor
X = [0:(length(A)-1)] * (1/25000);    % use sample rate to derive time base for plot
plot(X,A);
```

Headerless integer files

Headerless integer files are read in MATLAB by applying a data conversion factor.

1. Write the SIGNAL time buffer to a headerless integer time file. The following will write time buffer 1 to headerless file IFILE in default 16-bit integer (/D) format. Default format has an offset of 0 and a gain of 32768 bits/10 Volts, regardless of native A/D converter.

```
>W T 1 /B /D
Filename: IFILE
```

2. Read the file into a MATLAB array. The following MATLAB commands will read the entire integer file IFILE into the array A and apply a conversion factor.

```
fid = fopen ('ifile','rb');
A = fread (fid,inf,'int16');
A = A / 3276.8;
```

Exporting floating point SIGNAL time and frequency files to MATLAB

SIGNAL time buffers which have been manipulated after acquisition, as well as all power spectra and spectrograms, should be transferred as floating point data, to maintain precision. Floating point files are written in actual Volts, not bits, so no offset or gain conversion is necessary. This section covers time and power spectrum files, which are stored as one-dimensional arrays. Spectrograms are stored as two-dimensional arrays and are discussed in a separate section below.

SIGNAL-header floating point files

1. Write the SIGNAL time buffer to a SIGNAL floating point time file:

```
>W T 1
Filename: RFILE
```

2. Read the file into a MATLAB array. The following MATLAB commands will read the entire floating point file RFILE into array A, using FSEEK to skip the SIGNAL file header.

```
fid = fopen ('rfile','rb');
fseek(fid,1024,'bof');
A = fread (fid,inf,'float32');
```

3. **Example:** calculate the power spectrum of TWEET.1 in SIGNAL and store it as a SIGNAL-header floating point file, then read and plot this file in MATLAB. To display this file correctly, you must know its FFT length (8192 points) and sample rate (25000 Hz).

```
>R T 1                                  In SIGNAL
Filename: C:\SIGNAL\TWEET.1
>XF T 1 1
>W F 1
Filename: TWEET.F

fid = fopen ('tweet.f','rb');           In MATLAB
fseek(fid,1024,'bof');
A = fread (fid,inf,'float32');
X = [0:(length(A)-1)] * (25000/8192);
plot(X,A);
```

Headerless floating point files

1. Write the SIGNAL T or F buffer to a headerless floating point file. The following will write time buffer 1 to the headerless file RFILE in floating point (/F) format.

```
>W T 1 /B /F
Filename: RFILE
```

2. Read the file into a MATLAB array. The following MATLAB commands will read the entire floating point file RFILE into the array A.

```
fid = fopen ('rfile','rb');
A = fread (fid,inf,'float32');
```

Exporting MATLAB floating point time and frequency files to SIGNAL

MATLAB values are stored in floating point format, so all MATLAB data is exported as headerless files in 32-bit floating point format. When reading these files, SIGNAL will query the user to establish the time base of the data, since there is no file header to convey this:

Buffer type	Query parameters
T, F	Sample rate
FT	Sample rate, time range, no. time bins

1. The following will write the entire one-dimensional array A to the headerless floating point file RFILE:

```
fid = fopen ('rfile','wb');
fwrite (fid,A,'float32');
```

2. Read the file into a SIGNAL buffer. The following will read the floating point (/F) file RFILE into T buffer 1, assigning a sample rate of 25000 Hz:

```
>R T 1 /B /F
Sample rate: 25000
Filename: RFILE
```

Exchanging spectrogram files between SIGNAL and MATLAB

Spectrogram exchange requires special attention because spectrograms are two-dimensional objects and can be stored in multiple ways. The target program must know two things in order to correctly unpack a spectrogram file: 1) whether the data is stored by rows or by columns and 2) the actual dimensions of the spectrogram rows and columns.

Row-Column Order

MATLAB and SIGNAL write spectrogram files in the same order – ascending frequency at the earliest time value, then ascending frequency at the next time value, and so on. Thus the following spectrogram, consisting of 3 rows and 4 columns,

```
Freq  |  9   10  11   12
      |  5    6   7    8
      |  1    2   3    4
      -------------------
                Time
```

will be written by both programs in the following order in the file RFILE

```
1  5  9  2  6  10  3  7  11  4  8  12
```

Note that when displaying this matrix numerically, MATLAB will display the first row at the top of the screen, as shown below. However, when displaying the same matrix as a spectrogram, the first row (lowest frequency) will appear at the bottom of the graph. It should not be necessary to rearrange the stored matrix; this is a difference in display conventions.

```
1   2   3   4
5   6   7   8
9  10  11  12
```

Row-Column Dimensions

When importing into MATLAB, matrix dimensions are included in the FREAD command. When importing into SIGNAL, matrix dimensions are supplied to the read command via user queries. See the examples below.

Transferring Spectrogram Files from SIGNAL to MATLAB

Example: calculate the spectrogram of TWEET.1 in SIGNAL and store it as a SIGNAL-header floating point file, then read this spectrogram into MATLAB. To unpack this file correctly into a MATLAB array, you must know the number of FFT's and the number of points stored per FFT, and to display it you must also know its sample rate. These quantities are displayed by the SIGNAL BD command as NTIM, NFRQ, and SRATE, respectively. See the SPECTRO.M demo below to display the spectrogram matrix in MATLAB.

```
>R T 1                                      In SIGNAL
Filename: C:\SIGNAL\TWEET.1
>SET XFTLEN 256
>SET XFTSTP 100
>XFT T 1 1
>W FT 1
Filename: TWEET.FT

fid = fopen ('tweet.ft','rb');              In MATLAB
fseek(fid,1024,'bof');
A = fread (fid,[104,100],'float32');        % no. pts/FFT, no. FFT's
```

Transferring Spectrogram Files from MATLAB to SIGNAL

Example: store a spectrogram file from MATLAB, then read it into SIGNAL. To read the file into SIGNAL, you must know the number of FFT's, time duration, and sample rate.

1. In MATLAB, write the matrix array A to the headerless floating point file RFILE:

```
fid = fopen ('rfile','wb');
fwrite (fid,A,'float32');
```

2. In SIGNAL, read the floating point (/F) file RFILE into FT buffer 1. The command queries establish the FT buffer dimensions and time base.

> **`>R FT 1 /B /F`**
> Sample rate: **`25000`**
> Time range (msec): **`250`**
> No. time steps: **`100`**
> Filename: **`RFILE`**

Exporting RTS files into MATLAB

The RTS writes and reads only time files. It can write either SIGNAL-header or headerless files, but can read only SIGNAL-header files. Therefore the RTS can export time signals to MATLAB as SIGNAL-header time files but cannot import them from MATLAB.

MATLAB Demo Files

SIGNAL provides two MATLAB M-files in \demos which demonstrate some of the above techniques. TFILE.M demonstrates file import and export commands, and SPECTRO.M displays two- and three-dimensional spectrograms in MATLAB, based on a spectrogram file imported from SIGNAL. These demos use three sound files provided in the \sysdata directory: tweet.1, tweet.f, and tweet.ft. SPECTRO could be the basis for a general spectrogram display routine. **Note:** these files were tested with MATLAB version 5.2.

MATLAB Sound Processing Functions

MATLAB version 5.2 offers several sound processing functions, notably the ability to read and write Wave files (WAVREAD and WAVWRITE) and to play sampled sound data through the PC speaker (SOUND and SOUNDSC). These functions have the following limitations.

1. WAVREAD and WAVWRITE handle data only between -1 and +1, truncating any sample values outside this range. Because SIGNAL's acquisition range is ±10 Volts, these functions are not recommended for data exchange with SIGNAL.

2. Like all software that plays through the PC's built-in "sound card" hardware, SOUND and SOUNDSC are restricted to the available hardware sample rates, namely 44100 Hz, 22050 Hz, etc. To accommodate source signals of any sample rate, these functions automatically resample the playback signal mathematically to the nearest available hardware rate. This will usually introduce significant distortion, and so these functions are recommended only for signals with PC-compatible native sample rates.

Issues in Data Storage and Exchange

This section provides background on data storage issues that affect data exchange between software systems and hardware platforms. A basic understanding of these issues is important to exchange data accurately and reliably.

Text vs. Binary format

Data files can be stored and exchanged in either text (ASCII) or binary format. **Text format** is the most universal in that virtually all hardware platforms and software systems can read it, however it imposes two limitations: 1) it requires considerably more storage (up to 10 times more) for the same data and 2) numerical precision is limited to the data format chosen at the time of writing (for example, F12.4 format limits precision to 4 decimal places). By contrast, **binary format** is more space-efficient and numerical precision is determined by the data storage format inside the computer, which is then maintained in the data file. For these reasons, precision transfers of numerical data should usually be performed via binary files.

Integer vs. Floating Point format

Binary data in turn can be stored in either integer or floating point (real) format. **Integer format** stores the (integer) data in direct binary form, e.g., 1001 for the number 9. It has complete accuracy but a limited numerical range - e.g., a 16-bit integer can only represent integers between -32768 and +32767. **Floating point format** refers to the coding of real numbers, i.e., numbers which can assume non-integer values such as 1.234567. In this format, a certain number of bits (the "mantissa") represent the numerical value without regard for decimal point (e.g. 1234567 in the above), while the remaining bits (the "exponent") indicate the position of the decimal point (i.e. 10 ** -6 in the above example). The number of mantissa bits limits the precision of the format, while the number of exponent bits limits its dynamic range (largest positive and negative number). IBM PC's use a standard 32-bit floating point representation (IEEE 754) which has 1 sign bit, a 24-bit mantissa (1 bit is implicit), and an 8-bit exponent. This provides a mantissa resolution of 2 ** 24 = 10 ** 7, or 7 decimal places, and an exponent range of 2 ** 8 = 128 (i.e., the binary "decimal point" can be moved up to 128 binary places in either direction). Converting to powers of ten, the dynamic range is then 2 ** ±128 = 10 ** ±37.

The two numerical formats serve quite different purposes. Raw acquisition data is normally maintained in the original integer format delivered by the digitizer, to avoid introducing even minor "noise" by converting the integers to the less precise floating point format. Conversely, the results of calculations are almost universally stored in floating point format, since their precision and range vary widely.

Data Precision

Another issue in data storage is different **precision levels** within each numerical format, determined by the number of bytes allocated to store each number in the computer. Obviously, more bytes allow for greater precision and numerical range. Many analysis systems (including SIGNAL) use data lengths of 2 bytes (16 bits) for integer and 4 bytes (32 bits) for floating point, as described above. However, most computers also provide "double precision" data types consisting of 4-byte integers and 8-byte floats. These are the default representations of MATALB, although they are rarely necessary with signals having a dynamic range of less than 100 db, which includes virtually all acoustic signals. The following table summarizes the precision and numerical range of these data formats.

Data Type	Precision	Range
2-byte integer		±32768
4-byte integer		±2,147,483,648
4-byte float	7 decimal places	10 ** ±37
8-byte float	14 decimal places	10 ** ±74

Data Byte Format

A final issue in data storage is the order in which the bytes representing a number are stored in different computer models. This section applies only to users who are running MATLAB on a **non-PC platform**, such as Sun or Macintosh, and can be ignored by those who are running MATLAB on a PC, i.e., the same hardware platform as SIGNAL.

Different computers store the bytes making up an integer or floating point number in different order in memory. **Little-endian** format stores the least significant byte first, i.e., at the lowest memory location, while **big-endian** format stores the most significant byte first. Thus the decimal number 1000 = 03e8 hexadecimal would be stored in 16-bit little-endian format as

```
        8               e               3               0
low address                    →                high address
```

and in big-endian format as

```
        0               3               e               8
low address                    →                high address
```

These formats are used by the following hardware platforms:

Little-endian	**Big-endian**
IBM PC	Macintosh
VAX	NexT
DEC Alpha	Sun
	IBM RS6000
	SGi Iris

To transfer data into or out of MATLAB to a hardware platform using a different byte-order, use the optional FORMAT parameter in the FOPEN statement to specify the byte-order order of the incoming or outgoing data. MATLAB will then perform byte conversion as necessary. For example, to import data from SIGNAL on the PC to MATLAB on a non-PC platform, open the data file in MATLAB using the following (the field stands for IEEE format, little-endian, the byte format of PC data). MATLAB will then convert the data from this format to the format of its native machine.

```
fid = fopen ('myfile','rb','ieee-le')
```

Similarly, to export data from MATLAB on a non-PC platform to SIGNAL on a PC, have MATLAB store the data in little-endian IEEE format (ready for use by SIGNAL on the PC) by including the following:

```
fid = fopen ('myfile','wb','ieee-le')
```

4-4-99
Rev 1-15-04