

ALL ABOUT DIGITRAX SOUND DECODERS

By Fred Miller, MMR

PAGE

INTRODUCTION

| | |
|---|---|
| THE ROLE OF SOUND IN MODEL RAILROADING..... | 2 |
|---|---|

SECTION I - BASICS

CHAPTER

| | | |
|-----|---|----|
| I | DIGITRAX SOUNDFX® DECODERS..... | 4 |
| II | DIGITRAX SOUNDLOADER® SOFTWARE..... | 6 |
| III | SOUND PROJECT FILE (SPJ) | 8 |
| IV | SIMPLE MODIFICATIONS TO THE SOUND PROJECT FILE..... | 9 |
| V | CV AND FKEY ASSIGNMENTS IN A SOUND PROJECT | 13 |

SECTION II – MORE INVOLVED

| | | |
|------|---|----|
| VI | MORE COMPLEX MODIFICATIONS TO SOUND PROJECTS..... | 14 |
| VII | MODIFYING SOUND FILES IN A SOUND PROJECT..... | 17 |
| VIII | INTRODUCTION TO THE SDL MACRO LANGUAGE..... | 20 |
| IX | USING THE MACRO ASSEMBLER..... | 24 |

ALL ABOUT DIGITRAX SOUND DECODERS

INTRODUCTION

The scope of our model railroading has continued to grow over the many years since hobbyists first started modifying their tinplate trains. Modelers have been building more detailed representations of the prototype railroads that they reproduce in miniature. In the past dozen years or so capability has been made possible by better building materials and the application of advanced electronics. The introduction of Digital Command Control (DCC) is one example of a capability that enables more realistic operation of model railroad equipment.



FREIGHT OPERATIONS ON AUTHORS TRACTION LAYOUT

Availability of relatively inexpensive miniaturized electronics producing sounds has also brought a whole new dimension to the model railroad hobby. Although sound in the form of whistles and horns was introduced many years ago in the tinplate train sets, recent advancements enable the use of exact sound representations appropriate to specific models. Electronic equipment is now available for “modeling” the sounds of all types of steam and diesel locomotives, interurban and streetcars, and even rolling stock with unique sounds such as cattle cars and refrigerator cars.

The application of sounds on the model railroad layout is not limited just to the operating trains. Whole scenes can be created with background sounds to bring the layouts into a multi-dimensional model. Commercial units for “off-track” sounds are available from such manufacturers as MRC (Model Rectifier Corp), ITT (Innovative Train Technology Co.), RamTrack (Ram Radio Controlled Models Inc.). These offerings are self-contained electronics units that play unique sounds varying from crickets and farm critters to city and industrial sounds. Small electronic sound playing units are available for a user to record and playback custom sounds. One very capable device called the Dream Player (Pricom Design) plays hours of user supplied sounds recorded on flash memory cards. To enhance the background sounds a large collection of sound recordings are available on CDs and the Internet.



AUTOMATED CAROUSEL WITH SOUND ON AUTHOR'S LAYOUT

ALL ABOUT DIGITRAX SOUND DECODERS

Similarly several manufacturers provide sound units to be installed in model railroad rolling stock. Some of these offerings are compatible with DC propulsion systems but most are designed for DCC where access to throttle function keys enables comprehensive control of the sounds supplied in the sound unit. DCC sound units, called decoders, are available from manufacturers such as Loksound, Custom-Traxx (Tsunami), MRC and Digitrax. Each manufacturer provides a range of steam and diesel sounds appropriate for specific equipment.

The Digitrax line of SoundFX® decoders is currently the least expensive solution to implement sounds in DCC model railroad equipment. In addition the Digitrax decoders offer great flexibility in customizing not only the sounds, but also the behavior of the decoder in reaction to various operating situations and throttle function keys. The rest of this document will focus on the Digitrax sound system decoders and supporting equipment.

To date Digitrax has published operating manuals for each of their sound decoders. Unfortunately a comprehensive “Sound Decoder Programming Manual” or a “Language Tutorial/Description” is not available. Users in an adhoc manner have developed the knowledge base in the expanded use and modification of the very capable Digitrax decoders. Much of the information has been shared through the Yahoo Digitrax Sound Users group. This document was developed to bring that information together as an aid to interested Digitrax sound decoder users.



INTERURBAN OPERATIONS ON AUTHOR'S LAYOUT




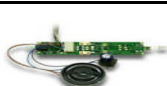

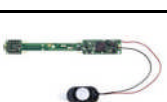







DIGITRAX SFX DECODER MOUNTED IN AUTHOR'S HO STREETCAR

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER I – OVERVIEW OF THE DIGITRAX SOUNDFX® DECODERS

Digitrax SoundFX® decoders are available in a variety of formats and pre-loaded sounds. Some decoders are “sound only” while others are combined motor and sound decoders. The following chart summarizes the current line of Digitrax Sound decoders:

| DIGITRAX DCC SOUND DECODERS | | | | | | | |
|---|------------------------------|-------------------------------|---------------------|--------|-----------------------------|-------------------------|------------------------------|
| Decoder | | Flash Memory Size | Speaker | Voices | Supported Sound Clip Format | Max Sound Clip Playtime | Total Decoder Sound Playtime |
| LEGACY DECODERS | | | | | | | |
|  | SFX004 (Sound Bug) | 4 Megabits (524,888 bytes) | 32 ohm 28mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
|  | SFX064D | 4 Megabits (524,888 bytes) | 32 ohm 28mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
|  | SFX0416 | 16 Megabits (2,097,152 bytes) | 32 ohm 28mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~3 min |
|  | SDH164xx series | 4 Megabits (524,888 bytes) | 32 ohm 28mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
|  | SDH164D | 4 Megabits (524,888 bytes) | 32 ohm 28mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
|  | SDN144xx series | 4 Megabits (524,888 bytes) | 8 ohm 13mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
|  | SDN144PS | 4 Megabits (524,888 bytes) | 8 ohm 13mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
| NEW SERIES-6 STANDARD DECODERS | | | | | | | |
|  | SDH166D (Replaces SDH164) | 4 Megabits (524,888 bytes) | 8 Ohm, 16x26x9 mm | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
|  | SDN136PS (Replaces SDN144PS) | 4 Megabits (524,888 bytes) | 8 ohm, 10x18mm oval | 3 | 8 bits, 11Khz, Mono | ~12 sec | ~¾ min |
| NEW SERIES-6 PREMIUM DECODERS | | | | | | | |
|  | SDXH166D | 16 Megabits (2,097,152 bytes) | 8 Ohm, 16x26x9 mm | 4 | 8, 12, 16 bits, 11Khz, Mono | ~ 95 sec | ~3 min |
|  | SDXN136PS | 16 Megabits (2,097,152 bytes) | 8 ohm, 10x18mm oval | 4 | 8, 12, 16 bits, 11Khz, Mono | ~ 95 sec | ~3 min |

ALL ABOUT DIGITRAX SOUND DECODERS

Some SoundFX[®] decoders, such as the “Board Replacement” types come with sounds specific to a prototype locomotive. Other SoundFX[®] decoders were made to fit a variety of existing DCC installations so they come with generic diesel and steam sound schemes. Some SoundFX[®] decoders include a 32 ohm speaker but you can substitute different speakers. Speakers, or combination of speakers, with as low as 8 ohms can be used successfully, however the overall volume may need to be reduced to prevent amplifier overload when using 8 ohms where the decoder was expecting 32 ohms. The latest decoder releases are using 8 ohm speakers, some with an enclosure. An external hold up capacitor is included to improve performance on dirty track. The speakers and the capacitor are wired external to the decoder proper to give flexibility in the installation of the decoder.

As is the case with all DCC decoders, the Digitrax sound decoders have a micro-controller (computer chip) central to their operation. In most cases the software program running in the micro-controller is a fixed series of operations based upon the parameters established in the form of Control Variables (CVs). The Digitrax SFX[®] decoders are unique in that it is possible to re-program the controlling software. This is a somewhat more complicated procedure than just replacing sounds but will be discussed later in this document.

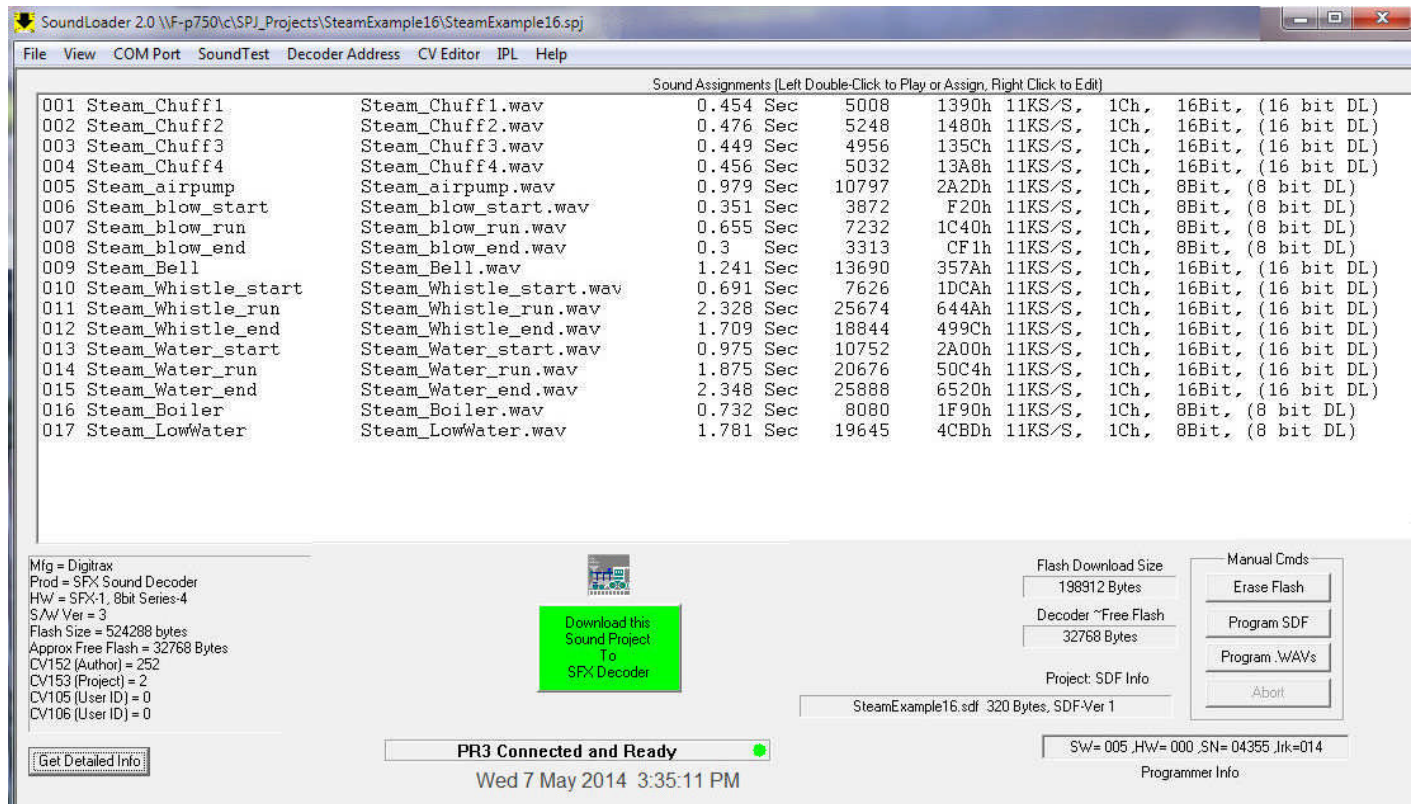
The “Legacy” and Standard Series-6 DigitraxFX[®] sound decoders are capable of three “voices” while the Premium Series-6 decoders have four “voices.” That means they can independently play up to three (or four) different sounds at the same time. The program running inside the decoder can initiate these sounds based on a number of internal or external activities, for example pressing a throttle function key or changing the locomotive speed. The sounds defined in that program have a priority for playing, i.e., within one of the voices, a higher priority sound could interrupt a lower priority sound. Care must be exercised in setting those priorities when building a new program.

In addition, the software running in the decoder could have multiple “schemes” or personalities. Only one is active at a time but can be selected with a Configuration Variable (CV) setting. Many of the original Digitrax SoundFX[®] decoders were released with a sound program of two schemes, one a generic diesel locomotive and the other a steam locomotive. Changing the Scheme Selection CV (CV60) changes the “personality” of the decoder from diesel to steam. Some recent Digitrax SoundFX[®] decoders, e.g., SDXH166D, are sold with 8 different sound schemes reflecting a variety of locomotives. Other sound programs developed by users provide variations in the operating characteristics and sounds by changing the Scheme Selection CV.

It is possible to customize a decoder's sound repertoire by using other sound recordings in standard Microsoft .wav format. Any locomotive sound part (a chuff, or a brake squeal, etc) can be replaced with an actual sound recording. This is done with the PC-based program SoundLoader[®] in conjunction with a Digitrax PR2 or PR3 programmer. This process is discussed in the next chapter. Additional information about construction of sound files is discussed in Chapter VII.

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER II – DIGITRAX SOUNDLOADER® SOFTWARE



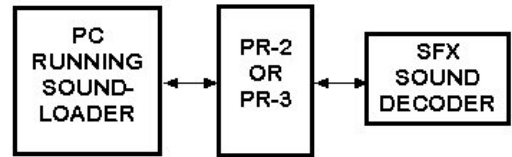
Digitrax provides a software program to use in modifying the sounds and/or program running in their SFX® sound decoders. This software is called SoundLoader® and the most current version (V2) is available on the Digitrax web site. It is also distributed with the PR2 and PR3 interface devices. SoundLoader's main function is to download Sound Project Files (.SPJ files) to the sound decoder in a locomotive on a programming track via a PR2 or PR3. The next chapter describes the contents of Sound Project Files (.SPJ file). SoundLoader® has other capabilities, too:

- ◆ The program is used to install new customized sounds in place of existing project sounds
- ◆ Once new custom sounds have been established on the SoundLoader® main screen, the sound project file (SPJ) can be saved with a new name, thus creating a 'custom' sound project file that can be shared with others.
- ◆ SoundLoader® can play the individual sounds through the operating PC speakers.
- ◆ In addition, the program can initiate the sounds in the connected sound decoder using an on-screen simulated throttle, complete with F keys, direction and speed controls.
- ◆ SoundLoader® is also handy for changing CVs in the connected sound decoder.
- ◆ The latest version of SoundLoader® (V2.0) can also “download” (DL) sound clips into the decoder in a lesser bit structure. For example download an original 16 bit sound clip as a 12 or 8 bit sound clip. This saves decoder memory at the expense of lesser quality sound..

ALL ABOUT DIGITRAX SOUND DECODERS

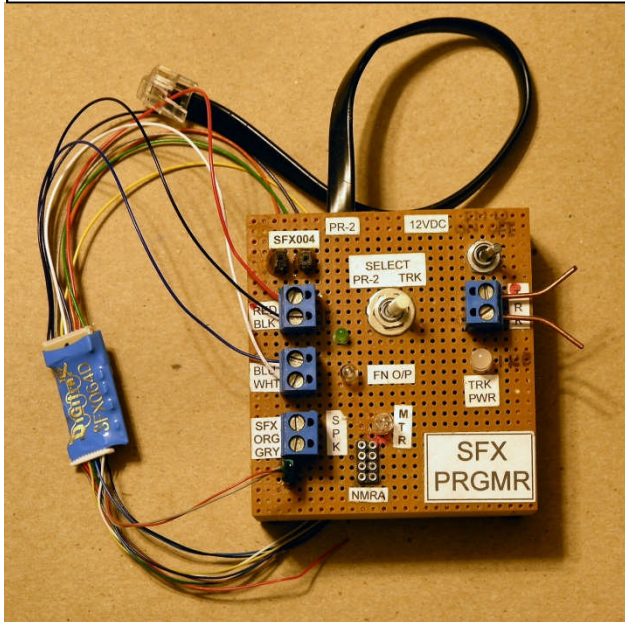
SoundLoader® runs on a Windows PC and connects to the sound decoder using the Digitrax PR2 or PR3 programmer and a programming track. The older PR2 programmer interface has 2 plug sockets, an RJ12 that connects the PR2 to the programming track and a 12V DC power supply (PS12 or PS14 -available from Digitrax), and a DB25 serial plug that connects to the PC.

A DB25 to DB9 connector is also included with the PR2 for PCs that only support the DB9 configuration. A USB adapter can be used with the PR2 if a serial port is not available.



The newer PR3 already has the USB connectivity through a USB cable to the Windows PC. 12 VDC supply is necessary (PS12 or PS14)

AUTHOR'S SFX DECODER MULTIPURPOSE PROGRAMMING RIG



ORIGINAL
DIGITRAX PR-2



DIGITRAX PR-3

ALL ABOUT DIGITRAX SOUND DECODERS

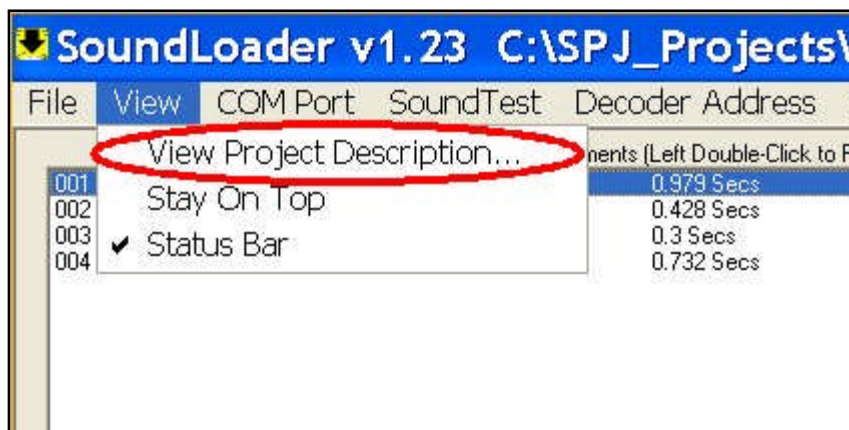
CHAPTER III - SOUND PROJECT FILE (SPJ)

As indicated earlier, the Digitrax Sound Project file (SPJ) is the heart of the operating control of the SFX sound decoder. HOWEVER, AN SPJ FILE CANNOT BE VIEWED OR DOWNLOADED FROM A SOUND DECODER. To see the components of a sound project, an original SPJ file must be loaded into the Digitrax SoundLoader® or other software such as the author's SPJVIEWER or SPJHELPER. SPJ files can be obtained from various sources including the Digitrax web site and the Digitrax Sound Yahoo user group. Construction of custom SPJ files will be discussed later in this document.

The SPJ "file" actually contains four different file types:

- (1) All of the actual sound clips (in a specific Microsoft WAV file format)
- (2) A MAP file that matches the sound clips (in the sequence shown by the SoundLoader®) to the identification in the SDF code. The MAP file from a SPJ can be exported using SoundLoader® and then viewed in a text editor like Windows NOTEPAD.
- (3) The SDF code, which makes it all run, is compiled into a HEX file from an Assembly Language (ASM) file using a Macro Assembler such as is available from MicroChip Technologies (and included in their free MPLAB IDE). Note this whole process actually develops the programming code, which subsequently runs in the micro controller inside the sound decoder.
- (4) A descriptive Text file (TXT) which serves two purposes:
 - (a) Readable documentation about the project
 - (b) Descriptive information, which pops up when using the SoundLoader® software, e.g., pop up descriptions of the F keys and the default CV values.

These Descriptive Files can be viewed directly in Sound Loader (VIEW pull down menu) and can also be exported for modification. Note that only the .TXT Descriptive file and the .MAP Handle file are normally accessible from an existing Sound Project using SoundLoader®. However a software product called SPJHELPER®, described later in this document, does provide a tool to extract these and the other components (WAV files, translated SDF file, etc.) from existing Sound Projects.



ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER IV – SIMPLE MODIFICATIONS TO THE SOUND PROJECT FILE

The SoundLoader[®] software, available without charge from the Digitrax website, is used to modify sound projects (SPJ) and download them into a decoder thru the PR2 (or PR3) Programmer. This program can be used without the PR2 (or PR3) to examine existing sample SPJ files and play the individual .wav sound files through the PC Speaker. SoundLoader[®] is used to manage a single "package" which contains the .wav sound files, SDF function code, a MAP file, and a TXT text description file.

The SoundLoader[®] display screen shows the following information:

SoundLoader 2.0 \\F-p750\\c\\SPJ_Projects\\SteamExample16\\SteamExample16.spj

FileViewCOM PortSoundTestDecoder AddressCV EditorIPLHelp

Sound Assignments (Left Double-Click to Play or Assign, Right Click to Edit)

| | | | | | | |
|-----|---------------------|-------------------------|-----------|-------|-------|---------------------------------|
| 001 | Steam_Chuff1 | Steam_Chuff1.wav | 0.454 Sec | 5008 | 1390h | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 002 | Steam_Chuff2 | Steam_Chuff2.wav | 0.476 Sec | 5248 | 1480h | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 003 | Steam_Chuff3 | Steam_Chuff3.wav | 0.449 Sec | 4956 | 135Ch | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 004 | Steam_Chuff4 | Steam_Chuff4.wav | 0.456 Sec | 5032 | 13A8h | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 005 | Steam_airpump | Steam_airpump.wav | 0.979 Sec | 10797 | 2A2Dh | 11KS/S, 1Ch, 8Bit, (8 bit DL) |
| 006 | Steam_blow_start | Steam_blow_start.wav | 0.351 Sec | 3872 | F20h | 11KS/S, 1Ch, 8Bit, (8 bit DL) |
| 007 | Steam_blow_run | Steam_blow_run.wav | 0.655 Sec | 7232 | 1C40h | 11KS/S, 1Ch, 8Bit, (8 bit DL) |
| 008 | Steam_blow_end | Steam_blow_end.wav | 0.3 Sec | 3313 | CF1h | 11KS/S, 1Ch, 8Bit, (8 bit DL) |
| 009 | Steam_Bell | Steam_Bell.wav | 1.241 Sec | 13690 | 357Ah | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 010 | Steam_Whistle_start | Steam_Whistle_start.wav | 0.691 Sec | 7626 | 1DCAh | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 011 | Steam_Whistle_run | Steam_Whistle_run.wav | 2.328 Sec | 25674 | 644Ah | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 012 | Steam_Whistle_end | Steam_Whistle_end.wav | 1.709 Sec | 18844 | 499Ch | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 013 | Steam_Water_start | Steam_Water_start.wav | 0.975 Sec | 10752 | 2A00h | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 014 | Steam_Water_run | Steam_Water_run.wav | 1.875 Sec | 20676 | 50C4h | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 015 | Steam_Water_end | Steam_Water_end.wav | 2.348 Sec | 25888 | 6520h | 11KS/S, 1Ch, 16Bit, (16 bit DL) |
| 016 | Steam_Boiler | Steam_Boiler.wav | 0.732 Sec | 8080 | 1F90h | 11KS/S, 1Ch, 8Bit, (8 bit DL) |
| 017 | Steam_LowWater | Steam_LowWater.wav | 1.781 Sec | 19645 | 4CBDh | 11KS/S, 1Ch, 8Bit, (8 bit DL) |

The sound "Number" and "Name", e.g., "007 Diesel_horn_begin" is listed in the sequence referenced in the MAP file. On the same line the actual wave file filename, play time and file size are shown. For example "Diesel_horn_begin.wav" is the original wave sound file name, which plays for 0.059 seconds and has a file size of 657 bytes. The 11025 sample rate, 1 channel (mono) and 8, 12, or 16 bits are standard for all SoundLoader[®] useable sound files.

A left mouse-click on any sound line will play that sound file through the PC speakers. A right mouse click on the sound line will pop up a menu to enable playing that sound in a loop (continuous until interrupted) or remove that sound from the project (substituting silence) or replacing the sound wave file with another sound wave format file.

Some sound files are used in sets. In the example above the three sounds "Diesel_horn_begin", "Diesel_horn_conf", and "Diesel_horn_end" would probably be played with the DCC Throttle F2 Function key as follows: "Diesel_horn_begin" plays once in it's entirety, followed by "Diesel_horn_conf" playing continuously (looping) while the F2 Function Key is held down, and then "Diesel_horn_end" would play once when the Function Key is released. Although the program running in the decoder controls the operation of these sounds, the actual sounds can be replaced using SoundLoader[®] software.

Creation and editing of sound files can be challenging but free software (e.g., Audacity and WavePad) is available to accomplish this task. Chapter VII is an extract from the Digitrax web site,

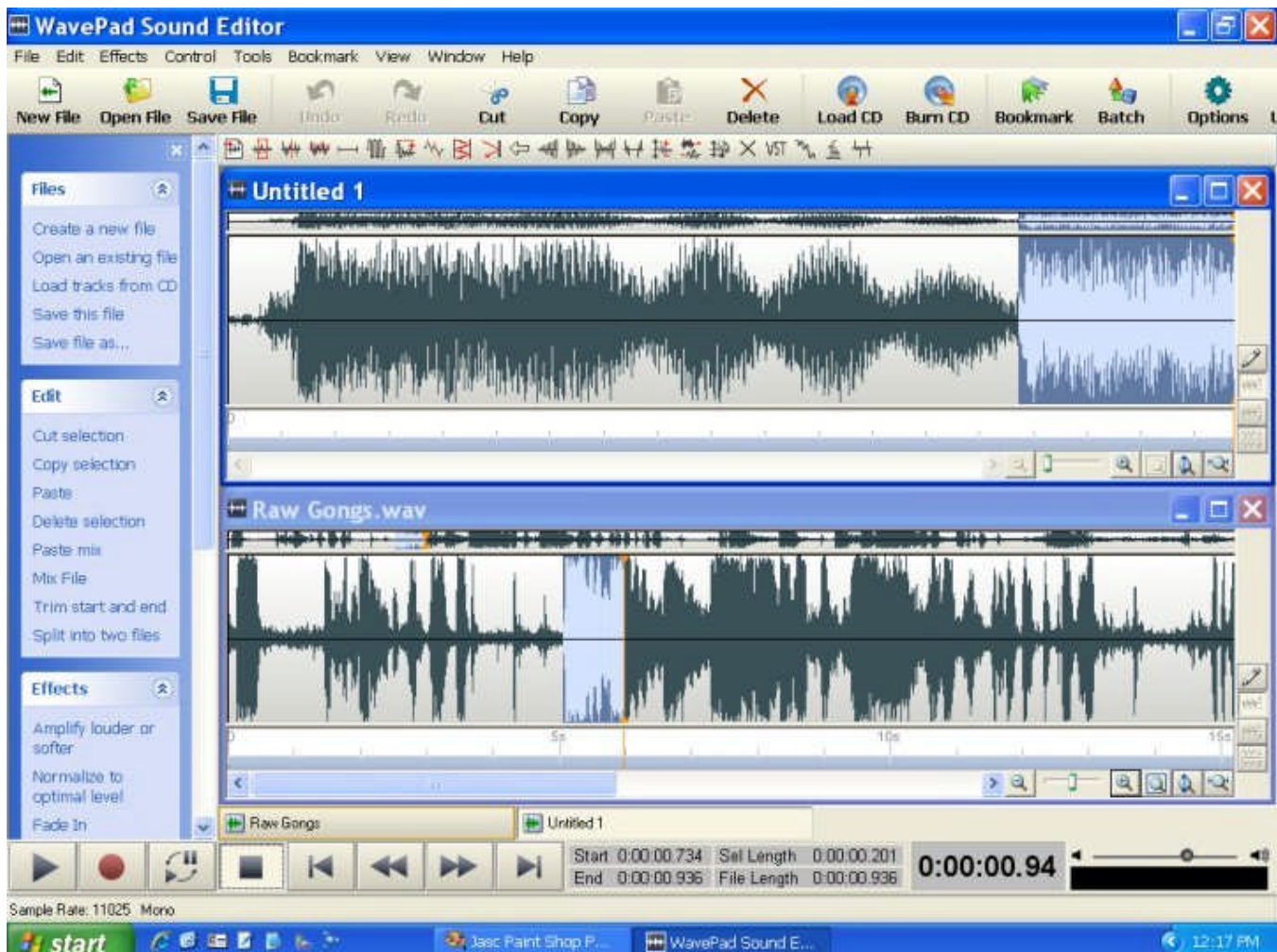
ALL ABOUT DIGITRAX SOUND DECODERS

which provides further guidance and suggestions on creating and editing sound files for use in Digitrax Sound Projects.

When replacing the sound files in the sound project, it is possible to erroneously put more .wav files into the project than the decoder can hold. When this happens, the decoder will not respond to movement instructions or play any sounds. The solution is to re-edit the sound project and remove some of the sound files and download it again. SoundLoader® does provide approximate information about the size of the project but it is easy to miss the fact that the result is more than the amount of decoder memory.

It is important to save any new .wav files in the correct format, always choose 11025 samples/second, mono and 8, 12 or 16 bits. The micro chip and the control software used in the Digitrax SoundFX® “legacy” or Standard decoders cannot address files greater than 128K (131,072 bytes or just shy of 12 seconds play time at the 11 MHz wave file sampling rate) If a file of greater length is loaded, the sounds for that clip AND all of the clips defined (in the sequential list) after that clip will not play. The newer Series-6 Premium decoders can handle individual sound clip files as long as 1,048,575 bytes, or about 95 seconds play time. However, sound clips defined earlier in the list will play, assuming the total memory size hasn't been maxed out.

ALL ABOUT DIGITRAX SOUND DECODERS



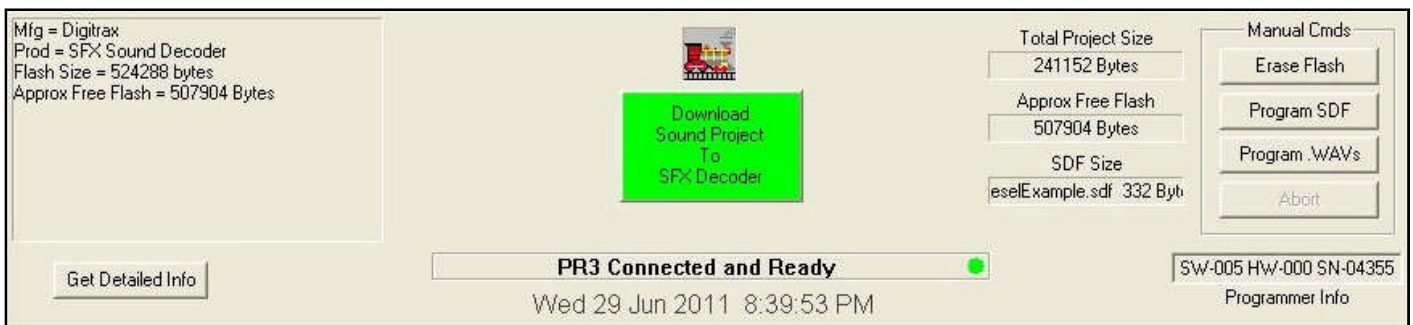
WAVEPAD is one of several free software programs that can be used to “edit” sound clips. Selecting portions of larger sound files and amplifying or otherwise changing characteristics are some of the functions that can be accomplished. The “cut and paste” operations are similar to word processing.

ALL ABOUT DIGITRAX SOUND DECODERS

TIPS:

- ◆ Don't use a file of greater than 131,072 bytes (Legacy/Standard decoders) or 1,048,575 bytes (Series-6 Premium) as shown in the SoundLoader® sound clip list)
- ◆ Make sure, after loading a project into a decoder, that the amount of "Available Free Flash" is NOT a negative number. Click on the "Get more info" button in lower left to assure a good reading. If it's negative, the project is too big.
- ◆ If you are having sound problems (and you don't have any of the above problems), try doing a "Manual Erase" of the decoder memory, cycle the power to the decoder (or PR 2/3), and then check to see if the "Available Free Flash" memory is the same as the total "Flash Size" shown above. Then do a new clean project load. The total size of the Sound Clips plus the SDF program are always a couple of K bytes larger than what SoundLoader® says is the "Project Size."
- ◆ Many of the Digitrax Sound decoders contain 4 Megabits of memory. That's 524,288 bytes of Flash Memory. The SFX0416 and the Series-6 Premium decoders contain 16 Megabits - or 2,097,152 bytes, four times as much Flash Memory for those 'bigger projects."

After a Sound Project's sound files have been adjusted as desired, a click on the SoundLoader's big green button titled "Download Sound Project to SFX Decoder" will copy the modified Sound Project (SPJ) into the decoder. The software will first erase any previous projects from the decoder and then download the new project.



If the big green button is not active, the screen will instead display diagnostic information which may indicate either the Software cannot access the PR3 (or PR2) because of an invalid or erroneous PC Port assignment, or a powered PR2/3 is not properly connected to a Digitrax sound decoder. The SoundLoader® HELP facility provides extensive guidance in proper connections and troubleshooting.

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER V – CV AND FKEY ASSIGNMENTS IN A SOUND PROJECT

Control Variables (CVs) are used to adjust the operation of a DCC decoder. A number of CVs are useful in modifying the operation of Digitrax Sound Decoders. These CVs can, among other uses, change the volume of some or all of the sounds, or even select between alternate sounds. Many of the sound decoder related CVs have a predefined function. However the function of other CVs in the range of CV140 - CV180 can have different functions depending upon the Sound Project loaded into the decoder.

IMPORTANT: THE LISTING OF CVS CONTAINED IN THE DOCUMENTATION SUPPLIED WITH DIGITRAX SOUND DECODERS APPLIES ONLY TO THE SOUND PROJECT INITIALLY LOADED INTO THAT DECODER. If other Sound Projects are subsequently installed, there MAY BE different CV assignments in the 140 - 180 range

In the same way, use of DCC Throttle Function Keys is defined in the Digitrax documentation for each new sound decoder. **THOSE ASSIGNMENTS MAY BE DIFFERENT FOR SUBSEQUENTLY LOADED SOUND PROJECTS.**

The Descriptive text file included in all valid Sound Project files (SPJ) should be consulted for CV and Function Key assignments. This Descriptive text file is accessible using SoundLoader® as described earlier.

Some of the CVs that have fixed functions are:

| | |
|-----------|---|
| CV11 | Code indicating Sound Time Out when loco address is de-selected |
| CV58 | Master volume setting (applies to all sounds) |
| CV60 | Selection of Sound Scheme (in multiple scheme projects) |
| CV132-134 | Various settings for Speed Notching, use of external CAM and Steam engine chuff gear ratios |
| CV135 | Volume setting when all sounds are “muted” (F8 or some other key defined in Sound Project) |
| CV140-180 | User defined. Although these are sound file dependent and can be user defined CVs, the SoundLoader® does reference CV152 & 153 for “author and project number display on the SoundLoader® screen. |

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER VI – MORE COMPLEX MODIFICATIONS TO SOUND PROJECTS

We saw earlier that the sounds played in the Digitrax sound decoders can be replaced or deleted in the Sound Project (SPJ) using the Digitrax SoundLoader® software. However more substantial changes and enhancements to the performance of the decoders can be accomplished by changing the software (program) within the SPJ. This capability is one of the features of that makes the Digitrax sound decoders excel.

As described earlier in this document, the Digitrax Sound Decoders are controlled by a micro-processor with appropriately loaded software. The software that controls the sound operations in the SFX decoders can be modified/replaced with new software written in a particular programming language. The Sound Definition Language (SDL) that AJ of Digitrax developed is used to build the code that runs inside the micro-controller (computer) in the sound decoders. As a result, fantastic capabilities are there for anyone who can jump into the pond.

Some of the possible changes to existing sound projects include reassignment of Function Keys, addition of new sounds, addition of new Function Key or motor related (speed, direction, etc.) sound activities and timer controlled sounds. Indeed whole new sound projects can be created for use in locomotives, rolling stock and off-line sound systems.

The MicroChip micro-controller, which Digitrax uses in their SFX decoders, could be programmed using all kinds of languages that are then compiled into the code that runs in the micro-controller (computer). However, at manufacturing time Digitrax loads some firmware (fixed software) into the decoders, which "interprets" a user (or Digitrax) supplied program. The latter program is written in the SDL language, which AJ created. It's really an ASM macro language, which means each statement in the SDL is broadened out to full ASM language. Two necessary "macro include" (.INC) files are used to do that "translation" of the user supplied SDL into code at assembly time using a Macro Assembler (MPASM). The resulting assembled code can then be loaded into the micro-controller (the .HEX file) using the Digitrax SoundLoader® software.

The Sound Project software (SDL) is written using the macro statements that reference the various "triggers" (Function Keys, Throttle changes, Timers, etc.) and sound clips of the Sound Project, to accomplish the desired sound performance. Other files are needed for the Sound Project as described in Chapter III. The software (in HEX format), the MAP and Descriptive Text files are all loaded into a new project within the SoundLoader® software. The MAP file will cause the SoundLoader® screen to display the sound clip names. Individual wave sound files must then be associated with the sound clip names in a manner discussed earlier in Chapter IV. If sound projects are to be developed using only text and sound editors and the Micro-Chip MPASM, it is highly recommended that extensive reference and copying from existing projects is made in order to get the correct formats. Installation and use of the Micro-Chip MPASM within their development system (MPLAB®-IDE) is beyond the scope of this document. Some guidance notes are provided in Appendix C. An overview of the SDL language is provided in Chapter VIII.

The SPJHELPER® software, available free from the author's website, offers a much simpler way to develop the software for the Digitrax SFX decoders. That software tool actually operates in two modes or personalities:

ALL ABOUT DIGITRAX SOUND DECODERS

- ◆ It has functional support for the *non-techie* to enable creation of all the components necessary for the Digitrax SoundLoader[®] software to load into a Digitrax SoundFX[®] decoder, and
- ◆ Alternatively, SPJHELPER[®] has easily accessible tools for the more *experienced programmer* to build the necessary files with ease.

In the first mode, users can create competent Sound Projects WITHOUT need for learning the SDL macro language, and indeed all references to the assembly process is simplified down to a mouse-click. Access to other supporting software such as a sound editor and the Digitrax SoundLoader[®] is also automated with simple mouse clicks. The SPJHELPER[®] automatically creates the necessary MAP files and supporting Descriptive text files necessary for the Digitrax SoundLoader[®].

Perhaps the most important feature of SPJHELPER[®] in the “simple” mode is the use of pull-down menus to select the desired functions. This completely eliminates the need to deal with the SDL macro language statements. A selection of “scripts” for more complex collection of activities is provided to further simplify the project development. Note however that the created ASM code is available to review or modify in the second “personality” of SPJHELPER[®].

As an example, the coding for a crossing horn is generated in SPJHELPER[®] with a few clicks of the mouse and selection of appropriate

```
E:0 Trigger when FKey 2 Turns On
Set Volume by value in CV 140
Play Diesel_horn_begin.wav
Play Diesel_horn_cont.wav
Play Diesel_horn_end.wav
Delay by 0.2 sec
Play Diesel_horn_begin.wav
Play Diesel_horn_cont.wav
Play Diesel_horn_end.wav
Delay by 0.2 sec
Play Diesel_horn_begin.wav
Play Diesel_horn_cont.wav
Play Diesel_horn_end.wav
```

sound files. SPJHELPER displays the generated coding in easily understood phrases.

The coding for that same function in SDL Macro

```
INITIATE_SOUND TRIG_SF7,NORMAL
LOAD_MODIFIER MTYPE_GAIN,IMMED_GAIN_MODIFY,SCV_140,SCALE_F
PLAY Diesel_horn_begin,no_loop,loop_STD
PLAY Diesel_horn_cont,no_loop,loop_STD
PLAY Diesel_horn_end,no_loop,loop_STD
DELAY_SOUND DELAY_THIS,8,DELAY_GLOBAL
PLAY Diesel_horn_begin,no_loop,loop_STD
PLAY Diesel_horn_cont,no_loop,loop_STD
PLAY Diesel_horn_end,no_loop,loop_STD
DELAY_SOUND DELAY_THIS,8,DELAY_GLOBAL
PLAY Diesel_horn_begin,no_loop,loop_STD
PLAY Diesel_horn_cont,no_loop,loop_STD
PLAY Diesel_horn_end,no_loop,loop_STD
DELAY_SOUND DELAY_THIS,8,DELAY_GLOBAL
PLAY Diesel_horn_begin,no_loop,loop_STD
PLAY Diesel_horn_cont,no_loop,loop_STD
PLAY Diesel_horn_end,no_loop,loop_STD
END_SOUND
```

Assembler language would be more difficult for a non-programmer to grasp:

The second mode of operation of SPJHELPER[®] provides easy to use references to all of the software necessary to support development of SPJ projects at the ASM level. Access to the assembler, text and sound editors and the Digitrax SoundLoader[®] is provided with simple mouse-clicks. Selection and naming of sounds and CVs is easily accomplished and automatically included in the final sound project loaded into the SoundLoader[®] software for downloading to the SFX decoder.

Those users interested in modifying/developing sound projects at the ASM level should make reference to the notes in Chapter VIII on the SDL language structure, and make ample reference to the sample sound projects included with SPJHELPER[®]. In fact a good starting point would be to develop a sound project in the SPJHELPER[®] “simple” mode and then study/modify the generated

ALL ABOUT DIGITRAX SOUND DECODERS

ASM code in the "Nittie Gritty" mode. This approach has the added advantage of letting SPJHELPER® generate all of the necessary sound file references and supporting files.

- ◆ Go through the entire development and decoder load with the "simple" mode FIRST. That sets up all of the necessary files and references needed for the sound project and SoundLoader®.
- ◆ Go back and make ASM changes/additions and reassemble in the "Nittie Gritty" mode and load the NEW HEX file into SoundLoader®, and then the decoder. Note that ONLY the new HEX file need be loaded to the decoder using the appropriate commands in SoundLoader®.
- ◆ CAUTION: DON'T go back and (re)generate the project in the "simple" mode. That will cause your ASM changes to get replaced and you would have to go back into "Nittie Gritty" mode and post the previous changes again.

The image shows the SoundLoader v1.23 application interface. The 'File' menu is open, showing options like 'New Sound Project...', 'Open Sound Project File...', 'Save Sound Project File...', 'Import .MAP File...', and 'Import .SDF File...'. The 'Import .SDF File...' option is circled in red. A red arrow points from a text box to this option. The text box says 'Actually Import the HEX file, not the SDF file'. Another red arrow points from a second text box to the 'Program SDF' button in the 'Manual Cmds' section of the main window. The second text box says 'Download HEX file to decoder, don't need to download whole project'. The main window displays project statistics: Total Project Size (26624 Bytes), Approx Free Flash (507904 Bytes), SDF Size (test.sdf 74 Bytes), and Manual Cmds (Erase Flash, Program SDF, Program .WAVs, Abort).

SoundLoader v1.23 C:\SPJ_Projects\tes

File View COM Port SoundTest Decoder Address CV E

New Sound Project... Ctrl+N
Open Sound Project File... Ctrl+O
Save Sound Project File... Ctrl+S
Save sound Project File As...
Import .MAP File...
Export .MAP File...
Import .SDF File...
1 C:\SPJ_Projects\test\test.spj
2 speedtesting.spj
3 DieselExample.spj
Exit

Actually Import the HEX file, not the SDF file

Download HEX file to decoder, don't need to download whole project

Total Project Size
26624 Bytes
Approx Free Flash
507904 Bytes
SDF Size
test.sdf 74 Bytes

Manual Cmds
Erase Flash
Program SDF
Program .WAVs
Abort

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER VII – MODIFYING SOUND FILES IN A SOUND PROJECT

This Appendix is an extract from Digitrax Web site Knowledge Base, used with permission from Digitrax

If you're really motivated you can take .wav files and make your own customized sound project files. This process of converting 'raw' sounds recorded in the field into a sound project file is exacting if the finished sound project file is to sound realistic. There are two kinds of sounds in SoundFX® decoders, simple sounds, and sequenced sounds. A simple sound is a sound that always sounds the same and always has the same length. An example of a simple sound is a bell. The striker strikes the bell and it rings for a certain length of time.

To prepare a simple sound, using any sound editor software (there are several links to free sound editors on the Sound Depot web page):

- ◆ Isolate the sound by trimming the excess time from the recording by carefully marking the beginning and the end of the sound you want to hear.
- ◆ Save this trimmed file as a .wav file. Make sure the saved .wav file format is "8 bit" and 11 KHz. Be sure to take note of where on your hard drive you saved this file.
- ◆ Open (run) the SoundLoader® software.
- ◆ The main screen of SoundLoader® will show you various parts of the locomotive's sound scheme (Diesel Bell, Diesel Brakes, etc.).
- ◆ Locate the sound type you want to change in this list and "right-click" on that entry - one of the menu choices will be "Assign Sound File" - Select this option.
- ◆ Navigate (browse) to the new .wav file you've created, select the file and click the 'Open' button.

You've now successfully modified your Project file. Using the File|Save command, save your customized sound project file (SPJ) with a new file name.

You can repeat the above steps and replace as many (or all) of the sound types as desired in the original sound project file. Once you've created this new sound project file you can download it directly to your Locomotive using SoundLoader's "Program" button, email it to a friend who has a similar Locomotive, or upload the sound project file to the Digitrax Sound Depot website for other users to have fun with.

The other type of sound SoundFX® decoders support is sequenced sound. A sequenced sound is a sound that is made up of three parts: an Attack sound, a Sustain sound, and a Decay sound.

- ◆ *Attack* is the starting sound
- ◆ *Sustain* is the running part of the sound
- ◆ *Decay* is the 'end' sound

An example of a sequenced sound is the horn. Blowing the horn for 15 seconds requires an Attack sound that *begins* the sequence, a Sustain sound that *prolongs* the sound for as long as desired, and finally *ends* with a Decay sound. In actual practice, file sizes for the beginning and end of the sound, the Attack and Decay, may possibly be larger than the Sustain because the Sustain is

ALL ABOUT DIGITRAX SOUND DECODERS

simply a small snippet of sound repeated as long as needed.

Examples of SoundFX® sequenced sounds are the Whistle, Water Pump and Horn. In the SoundLoader® main screen you'll see each of these sounds have a Start, Run, and End component. To prepare a sequenced sound you'll need to make 3 .wav files (The *Start*, *Run*, and *End* parts).

The finer points of making natural sounding sequenced sounds include:

- ◆ Recording several complete sound events (several complete whistle blows from start to finish, etc.).
- ◆ If possible, make these recordings each time varying distance from the Locomotive. If you're going out to a distant site to make the sound recordings, you might as well come back with more than one recording to choose from. A single recorded whistle blow may sound great by the siding, but may not sound as good once you listen to the recording at home; get several recordings from different distances and give yourself the latitude to choose the best.
- ◆ Making several recordings will also help you avoid picking up unwanted background noise on all of the recordings. Things like a car driving by or a bird chirping are NOT sounds that come out of real locomotives. If you make several recordings of each sound you are more likely to be able to find sound fragments that don't have these *extras*.
- ◆ Once you've chosen the best recording, you'll need to isolate the complete sound by trimming the excess time from the recording.
- ◆ The goal is to get just the *complete sound event* (whistle/horn blow/etc) with almost no sound before or afterwards on the recording. Once you've got it, save this file. Save a copy of it (with a different name) in a safe place on your hard drive.

The **Start** Part

Listen to your newly trimmed sound. Depending on the sound editor you're using, playing from the beginning you'll eventually be able to "see" a place on the timeline where the sound stabilizes. In other words, you'll be able to see the point on the timeline where the *tone* of the sound starts to remain constant. That's the point where you want to make your first cut. Save this file segment in the 8 bit / 11 KHz format. Be descriptive, if it's a horn recording name it something easy for others to identify like:

GMF7_horn_start.wav

The **Run** Part

This one is straightforward; depending on how long the total recording is most of it will be the stable "run" part of the sound. Copy a piece of this sound (typically less than 1 second long) to the clipboard and save it. It's a good idea to use the sound editor to listen to this sound running as a continuous loop to make sure that the looping transition sounds realistic. If you like what you hear, name it something consistent and descriptive like:

GMF7_horn_sustain.wav

ALL ABOUT DIGITRAX SOUND DECODERS

The *End* Part

By now you've probably got the hang of it. Mark the point in your recording just before the run sound starts to change, copy from this point to the very end of your sound clip. Save it naming it something like:

GMF7_horn_end.wav

You're now ready to overlay these newly created sound components (.wav files) into another sound project file to create a new sound project file. You can customize any sound project file, there's no limit to what you can create with Digitrax SoundFX®.

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER VIII – INTRODUCTION TO THE SDL MACRO LANGUAGE

Computers (including micro-controllers) operate by following steps outlined in a “program.” Each step in a program is executed and then the computer moves to the next step. Some programs (and computers) also allow for “interruptions” in the processing, causing the execution of a different series of steps. The SDL language operating in the micro-controller of the Digitrax sound decoders does just that. The flow of control is interrupted by a variety of “triggers.” For example when a throttle Function Key is pressed, the program reacts to that “trigger” to perform the steps defined for that Function Key.

The program and micro-controller provide for variables or registers (memory locations) that can have values set and later tested. Some of these registers are defined to reflect the operational state of the decoder, e.g., current throttle setting (speed) or current value of a CV.

Construction (syntax) of code in the ASM file is important. The SoundLoader® and the decoder require the software statements to be organized in a specific pattern. The various sections of the software as well as all the action statements (program steps) are defined by SDL macros. A valid program would contain sections as follows:

- ◆ Names of Sound Clips (as will be referenced later in the code)
- ◆ Definition of any variables including names of CVs to be referenced later in the code
- ◆ Scheme and Voice section delineations (see below)
- ◆ Code within each Voice (within a Scheme)

The definition of the sound clips, CVs and any special user variables are placed early in the coding. The operating statements follow these definitions for each *SCHEME* and *VOICE* within each *SCHEME*. Note that Sound Projects can contain the code for a number of *SCHEMES* (*numbered from 0, 1, etc.*), only one of which is operational in a decoder, each selected by CV60 at run time. Also each *SCHEME* can have sound operations for up to three *VOICES*. That means 3 different sounds can be playing at the same time, one in each *VOICE*.

For development of new Sound Projects it is highly recommended that the overall syntax for a project be “copied” from an existing project. If using the “Nittie Gritty” mode of SPJHELPER®, a skeleton format can be automatically generated and subsequently fleshed out with code.

The code within a *VOICE* defines how the decoder will play sounds. One or more “events” are defined, each starting with a definition of what will “trigger” the following command activities. This triggering statement starts with an *INITIATE_SOUND* statement, followed with the name of the “trigger” command, followed by an indication of when the triggering should occur. Subsequent lines of code define actions to be taken such as setting volumes and playing specific sounds. Other actions could include setting values in various registers or testing the status of registers and branching control to other portions of code based on the test. The definition statements for a triggered event are terminated with an *END_SOUND* statement, (generally) the last statement in the event. ALL statements should be indented from the left margin (with one or more spaces or a tab) EXCEPT labels or Tags that are referenced in the “*BRANCH_TO*” command. The labels or

ALL ABOUT DIGITRAX SOUND DECODERS

tags must be in the left margin without indenting. Here's a simple example of a triggered event. Note the indenting for easy reading of code:

```
-----  
; F2 to play single "GONG"  
-----  
INITIATE_SOUND TRIG_SF2,NORMAL           ;triggered when F2 first turned ON  
  LOAD_MODIFIER MTYPE_GAIN,IMMED_GAIN_MODIFY,SCV_GONG_VOLUME,SCALE_F  ;set volume  
  PLAY HNDL_GONG,no_loop,loop_STD  
END_SOUND
```

Any text beginning with a semi-colon (;) defines a comment. Comments are suggested for easier reading and later reference. In this example the code segment will operate when Function Key F2 is first turned on. *TRIG_SF2* defines the Function Key and *NORMAL* defines "when first turned on. The first action command in this triggered event sets the volume for any following sound using a CV value. *SCV_GONG_VOLUME* is the name of the CV that would have been defined previously in the code. *HNDL_GONG* is the name of the sound clip also defined previously.

Note: The program only references the assigned sound clip name. That name is only associated with an actual sound clip wavefile in the SoundLoader[®] program prior to downloading the program to the sound decoder.

The next example adds more function to the above example by selecting different sounds depending upon a CV setting. Note that the added third statement initiates a comparison between a CV value and a program provided (*IMMED_DATA*) parameter. If the comparison result is equal (*SKIP_SAME*) the following statement is "skipped." That following statement is a branch to a label *PLAY_GONG_TWO*. The result of this combination of test and branch statements is that if a CV named *SCV_GONG_TYPE* currently has a value of "2" then sound clip *HNDL_GONG2* will play, otherwise *HNDL_GONG1* will play.

```
-----  
; F2 to play GONG2 if SCV_GONG_TYPE = 2, play GONG1, otherwise  
-----  
INITIATE_SOUND TRIG_SF2,NORMAL  
  LOAD_MODIFIER MTYPE_GAIN,IMMED_GAIN_MODIFY,SCV_GONG_VOLUME,SCALE_F  
  MASK_COMPARE SCV_GONG_TYPE,IMMED_DATA,2,COMP_7LSB,SKIP_SAME  ;If 2 skip Branch  
  BRANCH_TO PLAY_ONE  
  PLAY HNDL_GONG2,no_loop,loop_STD  
END_SOUND  
PLAY_ONE  
  PLAY HNDL_GONG1,no_loop,loop_STD  
END_SOUND
```

Note that the label, or tag, (*PLAY_ONE*) referenced in the *BRANCH_TO* is NOT indented. By convention the *INITIATE_SOUND* and the *END_SOUND* statements are indented somewhat (3 spaces) and the contained action statements are indented some more (6 spaces). Also note the placement of spaces and commas (,) within the statements to separate the command and the various parameters.

ALL ABOUT DIGITRAX SOUND DECODERS

Some other action “triggers” which are available in the SDL language include:

| | |
|-------------------------------|--|
| TRIG_SF1 through TRIG_SF28 | Function Keys |
| T_SPD_ACCEL1 | Throttle setting just caused acceleration |
| T_SPD_DECEL1 | Throttle setting just caused deceleration |
| T_SPD_IDLE | Throttle entering Idle state |
| T_SPD_IDLEXIT | Throttle leaving Idle state |
| TRIG_MOVING | Throttle has loco moving (not 0 speed) |
| T-SPD_DIR_CHNG | Throttle changed loco direction |
| TRIG_TIME_16PPS | This trigger is activated 16 times per second |
| TRIG_FACTORY_CVRESET | When CV8 is set to 8 or 9 to reset CVs |
| TRIG_SCAT0 through TRIG_SCAT7 | Available TIMERS which can have trigger time set by program |
| TRIG_DISTANCE | Execute when DISTANCE register counts down to 0 |
| TRIG_CAM | Execute when “white wire” is grounded, as in steam engine cams |
| TRIG_SND_ACTV11 | Executes when decoder is selected (or dispatched) |

The last parameter in an *INITIATE_SOUND* statement defines when the following statements will be executed. Possible parameters could be:

| | |
|----------------|--|
| NORMAL | execute once when the defined trigger first turns on, as in above examples |
| RUN_WHILE_TRIG | execute whole event continuously while trigger is on |
| NOT_TRIG | execute when the defined trigger turns OFF |

Some available action command statements are:

| | |
|-----------------|--|
| PLAY | Initiate playing a defined sound clip |
| LOAD_MODIFIER | Change value in a Register |
| MASK_COMPARE | Compare a Register to another Register or supplied value |
| BRANCH_TO | Change operation sequence to a named label (or Tag) |
| DELAY_CV | Pause execution for time defined in Register, e.g., CV |
| DELAY_THIS | Pause execution for time defined by a supplied value |
| SKIP_ON_TRIGGER | Like a comparison but looking at a Trigger state on or off (True or False) |

The *PLAY* command statement uses a parameter (in addition to the named sound clip) to define single or repetitive playing:

| | |
|---------------------|--|
| No_loop,loop_STD | means play the sound clip once to completion |
| Loop_till_init_TRIG | means play the sound clip repeatedly until the initiating Trigger is OFF – useful for holding a sustained sound such as a whistle or air pump. |

The *LOAD_MODIFIER* command uses a parameter to specify information about the next following sound clip or specify how a register may be changed.

| | |
|----------------------------------|---|
| MTYPE_GAIN | Set the volume (using a CV) |
| MTYPE_PITCH | Set the pitch (depending upon throttle speed setting) |
| MTYPE_BLEND | Blend the next to play sound into the currently playing, if any |
| MTYPE_SCATTER | Set the specified Timer |
| MTYPE_SND CV | Set a sound CV value |
| MTYPE_WORK_INDIRECT Parameter | Modify Register specified in first Parameter with CV Register in second |
| MTYPE_WORK_IMMED | Modify Register specified in first Parameter with value specified |

ALL ABOUT DIGITRAX SOUND DECODERS

The *MASK_COMPARE* statement can compare internal Registers with other registers (e.g. CVs or user defined work areas) or with constants supplied within the compare statement.

| | |
|-------------|--|
| IMMED_DATA | Compare first argument (Register) with supplied value in second argument |
| TARGET_DATA | Compare first argument (Register) with second argument (Register) |

The *MASK_COMPARE* statement also specifies a “bit mask” to select which bits of the Register or value are to be compared. The masks can be HEX or DEC values, or predefined names of values. Normally a *COMP_ALL* is used to compare all 8 bits but a *COMP_7LSB* will use lower 7 bits, ignoring any +/- sign. Sometimes other masks may be needed to get at specific bits. An example is the Notch Register (WORK_NOTCH), which uses a different set of bits. The SND_CMD.INC file defines a number of useful masks and the name of the mask can be used instead of an actual HEX or DEC variable.

The last parameter of the *MASK_COMPARE* statement indicates what comparison is to be used:

| | |
|-----------|--|
| SKIP_SAME | If both arguments are equal, the next statement is <u>skipped</u> |
| SKIP_LESS | If first argument is less than the second, the next statement is <u>skipped</u> |
| SKIP_GRTR | If the first argument is equal or greater than the second, next statement <u>skipped</u> |

Some of the internal pre-defined useful Registers which can be accessed in the SDL code are

| | |
|-----------------------------------|--|
| WORK_SPEED | Current Throttle speed setting |
| WORK_NOTCH | Current Throttle Notch setting (0-7) |
| WORK_DISTANCE | Count down Register |
| WORK_USER_0 (through WORK_USER_5) | Memory Registers for containing Users values |
| MTYPE_SND CV | Set a sound CV value |

The above description of triggers, and action statements is just an introduction. A careful review of the comments and data in the SND_CMD.INC file will present additional Registers, Commands and Parameters. Study of the ASM code in the SPJHELPER® sample projects will give further insight into the SDL language.

ALL ABOUT DIGITRAX SOUND DECODERS

CHAPTER IX – USING THE MACRO ASSEMBLER

Note: Much of the complexity of using the Macro Assembler is eliminated through the use of SPJHELPER[®] as described earlier in this document. Installation and use is completely automated and “under the covers”

As noted in Chapter VI, the sound project software is written using predefined macro language. The program must be compiled, along with the .INC files, using a macro assembler. A capable Assembler is available without charge from the MicroChip web site. It is included in their Micro controller Programming Lab – Integrated Development Environment (MPLAB[®]-IDE)

Installing the Macro Assembler: The MPLAB[®]-IDE can be downloaded from MicroChip’s website: <http://www.microchip.com> Follow the links to download the latest copy into a temporary folder and then unzip all of the files into that temporary folder. Then run “Install_MPLAB_Vxxx.exe” (where xxx will be the latest version number.) When installing the software it may ask what devices you will be using. The Digitrax Sound Decoders use the 8-bit MCUs. You should specify that you want to use the MPASM suite. After the programs are installed you will have to do a restart.

Using the MPLAB[®] - IDE: It is useful to establish a folder with all of your project files in one place. This would include all of the MPLAB[®]-IDE files specific to your project as well as your .ASM program file and the Digitrax .INC files. (If this is your first project, it would useful to load an available .ASM project to get a head start with the required statements.)

1. After starting the MPLAB[®]-IDE use the *Project Wizard* in the Project Pull Down Menu. It will ask for your device which is the PIC18F242. When asked specify the project files (.ASM and the Digitrax .INC files)
2. When the Project Wizard finishes, the Source and Header files will show. A double click on the .ASM filename will open up an editor from which you can modify the source code.
3. Be sure to unselect the “Generate map file” checkbox in the Build Options of the Project Pull Down menu. The MPLAB[®]-IDE would otherwise make a .MAP file which would replace your Digitrax defined sound .MAP file. (This is an unfortunate use of the filename .map extension. The two files contain very different information.)
4. When you have the source file, as you would like, you can try an assembly by clicking on the *Make* command in the Project Pull Down menu. (There is also a MAKE icon in the tool bar.) If no errors are detected, you will see a BUILD SUCCESSFUL, otherwise you will see the errors listed. Corrections are necessary to the source code and then another MAKE until no errors are shown. (It is easy to switch back to the listing of the source by clicking on the [x] in the status window.

The MPLAB[®]-IDE will automatically save all changes to your .ASM and when you restart MPLAB[®]-IDE at another time, the same project will be loaded.