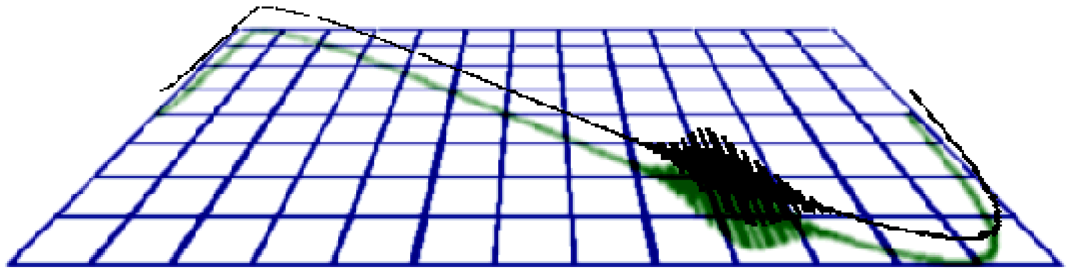
Out Of Phase

Version 1.1

Reference Guide



Thomas R. Lawrence

$\int_{I}^{P}$ Integral Productions dx

Table of Contents

License Agreement and Copyright

such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among

Acknowledgements

Several people helped in the design and construction of this program:

Craig Peeper showed me all kinds of awful file formats and described how many of the programs for MS-DOS/Windows worked. He also showed me how the Gravis UltraSound worked and gave me an efficient interpolation formula.

Eric Oehler helped design the synthesizer and provided insight into synthesis algorithms and systems used by the ‘pros’.

William Annis supplied information about synthesis mechanisms and psychoacoustics.

Matt Lawrence for interesting ideas for effects, and showing me how the Korg 01/W works.

Reporting Bugs

Few programs are ever free of bugs when first released. This is especially true for programs written by people with few resources. It is therefore possible that you may encounter bugs when using *Out Of Phase*. Please report bugs for everyone's benefit. I will make every effort to repair flaws in the program, especially ones that are potentially destructive.

The best way of sending a bug report is via e-mail to:

tomlaw@world.std.com.

If you can't send e-mail, then send paper mail to the following address:

Thomas R. Lawrence
512 E. Clark St. Apt #25
Champaign, IL 61820

Here are some guidelines for helping me track down problems:

1. Can you reproduce problem (i.e. can you reliably cause it to happen if you so desire)? If so, *indicate exactly how you can cause the problem to occur*, in a step by step manner, starting with launching the program. If you are reporting by e-mail, attach a binhex or uuencoded copy of a document which exhibits the problem, if possible.

2. Make sure you are using the most recent version of the program. Be sure to report the version number of the program for which the problem occurred.

2. Try disabling or removing other software from the system and then try to make the problem occur. If you find some software that appears to cause problems with *Out Of Phase*, please send the title, version number, manufacturer name, and a brief description of the purpose of the software.

3. If you are able to do so, try to get the problem to occur with a version of the program that was compiled with internal debugging enabled. Report any error messages you see, and send me copies of any dump files the program creates.

4. What is your system configuration? Pay particular attention to the following details:

   A. What model computer and operating system are you using?
   B. How much real memory do you have?
   C. What additional products do you run along with *Out Of Phase*. Include system software, extensions, hardware add-ons, accelerators, etc.
   D. Are you using virtual memory, RAM Doubler, or Optimem?

5. Please report the problem, even if you can not make the problem occur again. *Describe as accurately as possible what you were doing when the problem occurred*. I may still be able to find the cause.

Introduction

Out Of Phase is a comprehensive system for editing and playing electronic music. It provides facilities for building waveforms, editing samples, constructing instruments, and creating scores. The results can be recorded to hard disk at any sound quality for subsequent processing or mastering to some other medium. The only requirements are a reasonably fast computer with digital audio output hardware and a large hard disk. No additional hardware or MIDI equipment is needed.

Out Of Phase produces sounds based on two synthesis techniques, sampling and indexed wave tables.

Sampling is a common synthesis technique available in most computer music systems. A digital recording of some sound is loaded into the computer and played back at different pitches and loudnesses. One problem with sampled synthesis the lack of variety for an instrument. There is no way of changing the timbre of the sound from one note to the next to achieve subtle effects. In addition, samples usually have a small range of pitches outside of which they sound unnatural.

Synthetic methods of waveform production allow the creation of more dynamic and controllable sounds. There are many systems in use, including filtering and FM synthesis. Synthetic methods require considerably more processing power than sampling. As a compromise, *Out Of Phase* provides *indexed wave table* synthesis. This synthesis technique uses an array of samples, each representing one period of some waveform. An index is used to select which wave period to play at any given moment. By varying the index over time, dynamic changes in the sound can be achieved with low computational overhead. The wave table can be constructed by any means, including algorithmically, and *Out Of Phase* includes a built in programming language for this purpose.

Most aspects of synthesis are controlled by envelope generators, which are composed of linear or exponential line segments of various slopes and lengths. Each note contains four parameters which can be used to adjust the duration or target amplitude of any line segment in the envelope, allowing the timbre to be changed over a range of notes. In addition, three release points are provided for handling sustained notes. Notes may be tied together, with the option of resetting the envelope generators or allowing them to continue uninterrupted from the beginning of each note.

Low frequency oscillators (LFOs) can be applied to the output of most envelope generators to provide additional effects. LFOs are used for producing tremolo and vibrato, and can be used to create several other effects as well.

Each instrument may have several oscillators, allowing the construction of complex waveforms for single notes.

Programming Language Reference

Introduction:

Out Of Phase contains an interpreter for a programming language that is flexible enough for algorithmically generating waveforms and implementing transformations. The style of the language is a combination of FORTRAN, Pascal, and Scheme.

In the description that follows, standard conventions for specifying language grammars are used. Brackets [ ] contain constructs which are optional and can be omitted. Braces { } contain constructs which can be repeated multiple times or omitted.

Data Types:

The following data types are provided in the language:

bool	logical value, either **true** or **false**
int	integer, with range at least -2147483647 to 2147483647
single	32-bit floating point
double	64-bit floating point
fixed	24-bit fixed point, with range at least -127 to 127
boolarray	array of boolean values
intarray	array of integers
singlearray	array of 32-bit floating point values
doublearray	array of 64-bit floating point values
fixedarray	array of 24-bit fixed point values

Only one dimensional arrays are supported. Array length is determined at run-time when the array is constructed (*see array constructor*).

Constants:

Numbers of various data types can be specified as follows:

bool	**true** or **false**.
int	An integer is a sequence of digits with no decimal point. Examples of integers are **3621** and **-623156**.
single	A single precision number is specified as a number with an optional decimal point and exponent specifier, followed by the character **s**. Examples of single precision floating point values are **-3.14s** and **672.167e-13s**
double	A double precision number is specified as a number with optional decimal point and exponent specifier, optionally followed by the character **d**. Examples of double precision floating point values are **3.14d** ad **-17.362**.
fixed	A fixed point number is specified as a number with an optional decimal point and exponent specifier, followed by the character **f**. Examples of fixed point numbers are **-31.3f** and **52f**.

Expressions:

An expression is any statement described below, as well as constant values specified above.

Compound Expressions:

A compound expression is a sequence of expressions enclosed in parentheses:

> **(** <expr> **;** <expr> **;** ... <expr> [**;**] **)**

The expressions are evaluated in textual order. The value of the compound expression is the value of the last expression in the list.

Variable Declarations and Array Constructors:

A variable is declared by the following statement:

> **var** *<name>* **:** *<type>* **=** *<initial-value-expression>*

The initial value of the variable is the result of evaluating the initial value expression. Arrays are constructed specially, with the following variant:

> **var** *<name>* **:** *<array-type>* **(** *<integer-expression>* **)**

The integer expression specifies the number of elements the array should be created with. All elements are initialized to zero. Array variables may be created using the first type of declaration if the initial value expression results in an array.

The scope of a declared variable extends from the point of declaration to the end of the immediately enclosing compound expression.

Conditional Statement:

A conditional statement is an if statement which evaluates one of several expressions based on the result of evaluating boolean tests.

> **if** *<bool-expr>* **then** *<body-expr>* {**elseif** *<bool-expr>* **then** *<body-expr>*} [**else** *<body-expr>*]

The **elseif** clause may be repeated zero or more times, and the **else** clause is optional. At most one body expression is evaluated, and the if statement returns the value of the body that was evaluated. The type of value returned by all body expressions must be the same. If no body expressions are evaluated, then the if statement returns zero.

Loop Statements:

A loop statement executes its body expression until a test expression meets some termination condition.

> **do** *<body-expr>* **while** *<bool-expr>*
> **do** *<body-expr>* **until** *<bool-expr>*
> **while** *<bool-expr>* **do** *<body-expr>*
> **until** *<bool-expr>* **do** *<body-expr>*

The **do..while** and **do..until** variants perform the test after the body expression, so they always evaluate the body expression at least one. The **while..do** and **until..do** variants perform the test first, so they may not evaluate the body expression. The value returned from the loop statement is the result of the last evaluation of the body expression or zero if it was not evaluated.

Assignment Statement:

An assignment statement stores a new value in a variable, replacing the old value.

set *<l-value-expr>* **:=** *<expr>*

The l-value is either a variable or an array element.

Array Element:

An array element can be accessed by the following:

<array-expr> **[** *<integer-expr>* **]**

The array expression is some expression that returns an array; usually this is the name of the array variable. The integer expression specifies which element of the array to return. The first element in an array is indexed by zero.

Resize Array:

The number of elements in an array can be changed with the array resize statement.

resize *<array-expr>* **to** *<int-expr>*

If the new number of elements is greater than the previous number of elements, then the new elements are initialized to zero. If the new number of elements is less than the previous number of elements, the elements on the end of the array are discarded. The expression returns the array as its value.

Error Message:

The error message statement allows the program to present a dialog to the user giving the option of continuing or aborting the program.

error *<string>* **resumable** *<bool-expr>*

The string is any text surrounded by double quotes. If the boolean expression is **true**, then the user can either choose to continue computation or abort it. If it is **false**, the user can only abort. If control returns to the program, the resulting value of the statement is the boolean **true**.

Print Statement:

The print expression statement prints either a string or the result of an expression to a special output window.

print *<string>*
print *<expr>*

No line breaks are printed unless the newline escape sequence **\n** is included in a string. Print statements return the boolean value **true**.

Sample Fetch Statements:

The following statements can be used to fetch data or attributes for some sample or wave table. Data may only be fetched from stored wave tables or samples; algorithmically generated ones can not be accessed. The general syntax is:

<keyword> <string>

The following keywords are allowed:

getsampleleft	Get the left channel fixedarray for a sample. This is only allowed for stereo samples.
getsampleright	Get the right channel fixedarray for a sample. This is only allowed for stereo samples.
getsample	Get the fixedarray for a sample. This is only allowed for mono samples.
getwavenumframes	Get the number of frames per period table for a wave table.
getwavenumtables	Get the number of period tables that define a wave table.
getwavedata	Get a fixedarray of data for a wave table. The array is linear with wave periods packed one after the other.

The <*string*> specifies which sample or wave table the data should be fetched from.

Arithmetic Expressions:

The general form of an arithmetic expression is as follows:

<*expr*> <*op*> <*expr*> {<*op*> <*expr*>}

The expression chain can be as long as desired. Operators associate left-to-right within their precedence levels, except exponentiation, which associates right-to-left. There is no short circuiting of operators; all arguments are evaluated before the operator is applied. The following table summarizes operators and their precedence levels (higher levels have higher precedence):

Precedence Level 1:	
bool **and** bool —> bool	logical and
int **and** int —> int	bitwise and
bool **or** bool —> bool	logical or
int **or** int —> int	bitwise or
bool **xor** bool —> bool	logical exclusive or
int **xor** int —> int	bitwise exclusive or
Precedence Level 2:	
scalar **<** scalar —> bool	less than comparison
scalar **<=** scalar —> bool	less than or equal comparison
scalar **>** scalar —> bool	greater than comparison
scalar **>=** scalar —> bool	greater than or equal comparison
scalar **=** scalar —> bool	equal comparison
scalar **<>** scalar —> bool	not equal comparison
Precedence Level 3:	
scalar **+** scalar —> scalar	arithmetic addition
scalar **-** scalar —> scalar	arithmetic subtraction
Precedence Level 4:	
scalar ***** scalar —> scalar	arithmetic multiplication
int **/** int —> double	imprecise integer divide
fractional **/** fractional —> fractional	decimal divide
int **div** int —> int	precise integer divide
int **mod** int —> int	precise integer remainder
scalar **<<** int —> scalar	shift left
scalar **>>** int —> scalar	shift right

Precedence Level 5:

scalar ^ scalar —> double	exponentiation

Unary Operators:

Unary operators are functions of one argument. They always take precedence over binary operators, but they are preceded by array subscript operations.

not bool —> bool	logical not
not int —> int	bitwise not
sin scalar —> double	sine in radians
cos scalar —> double	cosine in radians
tan scalar —> double	tangent in radians
asin scalar —> double	arc sine in radians
acos scalar —> double	arc cosine in radians
atan scalar —> double	arc tangent in radians
ln scalar —> double	natural logarithm
exp scalar —> double	natural exponentiation
bool scalar —> bool	typecast to boolean (arg != 0)
int scalar —> int	typecast to integer (truncate)
single scalar —> single	typecast to single
double scalar —> double	typecast to double
fixed scalar —> fixed	typecast to fixed
sqr scalar —> double	square
sqrt scalar —> double	square root
abs scalar —> scalar	absolute value
- scalar —> scalar	negation
sign scalar —> int	sign of number (-1, 0, or 1)
length array —> int	get length of array

Function Call:

A function can be called by specifying the name of the function followed by an argument list:

<identifier> **(** [*<expr>* {**,** *<expr>*}] **)**

The arguments are evaluated left to right and then the function is called. If the function has not been declared in the same module before the code being executed, then a prototype must be provided (*see function declarations*).

Type Promotion:

Values of types **int**, **single**, and **fixed** are automatically converted to values of type **double** when necessary. This occurs when an **int**, **single**, or **fixed** number is supplied as an argument to a function expecting a **double**, or when a binary operator is applied to both an **int**, **single**, or **fixed** and a **double**.

Function Declaration:

Function declarations have the following form:

func *<identifier>* **(** [*<identifier>* **:** *<type>* {**,** *<identifier>* **:** *<type>*}] **) :** *<returntype> <expr>* **;**

The final type of the function's expression must match the declared return type. The formal parameters specified in the argument list are are bound to values resulting from actual parameter expressions evaluated when the function is called.

Binding for scalar types is by value. Binding for array types is by reference with the exception that assignment of new arrays to the formal parameter does not replace the array in the actual parameter.

To enforce type checking, function prototypes are required. A prototype has the following form:

> **proto** *<identifier>* **(** [*<identifier>* **:** *<type>* {**,** *<identifier>* **:** *<type>*}] **)** **:** *<type>* **;**

A prototype can occur wherever an expression can occur except that a prototype can not be the last element of a compound expression.

Instrument Specification Reference

Introduction:

The following section describes the specification language used for building instruments. Note that nearly all statements in the specification language must be terminated with semicolons.

Instrument Definition:

An instrument is defined by specifying a list of parameters inside of an instrument declaration:

instrument (*<instrument-elements>* **)**

The following instrument elements are allowed:

loudness *<number>* **;** *optional (default is 1)*
Specifies the loudness of the sound. The normal loudness is 1. If multiple oscillators are used, then a value of 1/(number of oscillators) is suggested to normalize the volume relative to other instruments.

frequencylfo (*<lfo-elements>* **) ;** *optional, repeatable*
Specifies an LFO to be applied to the pitch of the sound. LFO modulation is applied to the pitch in the order the LFOs are specified in the instrument definition. If the LFO is in *linear* or *hertz* mode, then its value is treated as Hertz, if the LFO is in *exponential* mode, then its value is treated as octaves, and if the LFO is in *halfstep* mode, then the valu treated as halfsteps. Unlike other LFOs, the default modulation for the pitch LFO is halfsteps.

oscillator [*<identifier>*] **(** *<oscillator-elements>* **) ;** *required, repeatable*
Specifies an oscillator for the instrument.

trackeffect *<effect-specifier>* **;** *optional, repeatable*
Specifies an effect to be applied to the track signal (i.e. the sum of oscillators for all playing notes on the track).

LFO Specifier:

The LFO specifier describes the nature of the low frequency modulation applied to some value, such as pitch, loudness, or wave table index. The following is a description of its elements:

freqenvelope (*<envelope-elements>* **) ;** *optional (default is constant 0)*
This field specifies the envelope controlling the frequency of the low frequency oscillator, in Hertz.

ampenvelope (*<envelope-elements>* **) ;** *required*
This field specifies the maximum amplitude of the low frequency oscillator. The meaning of the number generated here depends on how the value from this LFO is used.

oscillator *<oscillator-type>* **;** *optional (default is signsine)*
This field specifies what kind of waveform the LFO generates. It may be one of the following (all periodic waveforms are shaped similar to a sine wave, that is, the highest point occurs during the first half of the period, and the lowest point occurs during the second half):

constant	The value 1. This oscillator ignores the frequency envelope and outputs the values obtained from the amplitude envelope.
signsine	A sine wave ranging between -1 and 1.
plussine	A sine wave ranging between 0 and 1.
signtriangle	A triangular wave ranging between -1 and 1.

plustriangle	A triangular wave ranging between 0 and 1.
signsquare *<number>*	A square wave ranging between -1 and 1. The number specifies the fraction of the waveform spent on transition edges.
plussquare *<number>*	A square wave ranging between 0 and 1. The number specifies the fraction of the waveform spent on transition edges.
signramp *<number>*	A ramp (sawtooth) wave ranging between -1 and 1. The number specifies the fraction of the waveform spent on transition edges.
plusramp *<number>*	A ramp (sawtooth) wave ranging between 0 and 1. The number specifies the fraction of the waveform spent on transition edges.
signlinfuzz *<number>* {**square** or **triangle**}	A randomly generated wave ranging between -1 and 1. The number specifies the seed for the Park and Miller random number generator. **Triangle** means points are linearly interpolated between; **Square** means no interpolation is used. The frequency of point generation is controlled by the frequency envelope.
pluslinfuzz *<number>* {**square** or **triangle**}	A randomly generated wave ranging between 0 and 1. The number specifies the seed for the Park and Miller random number generator. **Triangle** means points are linearly interpolated between; **Square** means no interpolation is used. The frequency of point generation is controlled by the frequency envelope.
wavetable samplelist (*<sample-list>* **) envelope (** *<envelope-elements>* **)**	A wave table selected from the sample list by pitch. The envelope controls the wave table index.

modulation *<modulation-type>* **;** *optional (default is additive)*

The modulation operator used for combining the LFO value with the value it is modulating. It can be one of the following:

additive	add the values together
multiplicative	multiply the values together
inversemult	multiply (1 - modulator) by the modulated value

exponential ; or **linear ;** or **hertz ;** or **halfsteps ;** *optional (default is linear)*

This specifies the mode of the LFO, how the values are combined. Linear combines the actual values together under the modulation operator. Hertz is synonymous with linear. Exponential takes the base-2 logarithm of the target value, combines it with the modulating value under the the modulation operator, and then takes the base-2 exponentiation of the result. Halfsteps is the same as exponential, except that the LFO oscillator output is divided by 12 first.

Oscillator Specifier:

The oscillator specifier describes the structure of an oscillator. The following fields are recognized:

type *<oscillator-type>* **;** *required*

This specifies the type of oscillator, either **sampled** or **wavetable**.

samplelist (*<sample-list>* **) ;** *required*

This specifies a list of samples or wave tables to choose from, based on the pitch of the note being played.

loudness *<number>* **;** *optional (default is 1)*

This specifies the loudness of the oscillator relative to other oscillators in the same instrument. Usually this is 1.

freqmultiplier *<number>* **;** *optional (default is 1)*

This specifies a value to multiply the note’s frequency by in order to obtain the frequency that this oscillator should play at.

freqdivisor *<integer>* **;** *optional (default is 1)*
This specifies an integer value to divide the note's frequency by in order to obtain the frequency that this oscillator should play at.

freqadder *<number>* **;** *optional (default is 0)*
This specifies a value to add to the note's frequency in order to determine what frequency the oscillator should play at. This addition is performed after the freqdivisor and freqmultiplier are applied.

loudnessenvelope (*<envelope-elements>* **) ;** *required*
This specifies the envelope that controls the loudness of the oscillator. The loudness value should stay between -1 and 1.

loudnesslfo (*<lfo-elements>* **) ;** *optional, repeatable*
This specifies an LFO to modulate the oscillator's loudness.

indexenvelope (*<envelope-elements>* **) ;** *optional*
This specifies the envelope controlling the index into the wave table. This can only be specified for wave table based oscillators. The index value should stay between 0 and 1.

indexlfo (*<lfo-elements>* **) ;** *optional, repeatable*
This specifies an LFO modulating the index into the wave table. It can only be specified for wave table based oscillators.

stereobias *<number>* **;** *optional (default is 0)*
This specifies some amount that is added to the note's stereo positioning value.

displacement *<number>* **;** *optional (default is 0)*
This specifies how many seconds late the oscillator should be started relative to the start of the note. If the envelope has any segments before it's origin, this must be set to an appropriate negative value in order to hear any output. For samples, this value adjusts the sample's origin point relative to the start of the note.

frequencylfo (*<lfo-elements>* **) ;** *optional, repeatable*
This specifies a pitch LFO to be applied to the pitch. The LFO behaves the same way as the one specified for the whole instrument, except that the effects are limited to this oscillator.

Envelope Specification:

An envelope contains two fields:

totalscaling *<number>* **;** *optional (default is 1)*
A scaling value which is multiplied by the output of the envelope generator.

points (*<point-list>* **) ;** *required*
This specifies the list of line segments which define the envelope.

The point list is composed of the following elements:

delay *<number>* { **level | scale** } *<number> <attributes>* **;** *required, repeatable*
This specifies one line segment in the envelope. The delay specifies the number of seconds over which the transition should take place. The **level** variant specifies the output level which the line should end at. The **scale** variant specifies what multiple of the initial level the line should end at. The attributes may be chosen from the following list:

sustainpoint *<integer>* *optional, globally once-only*
This sets the specified key-up point to sustain. This means that when the envelope generator reaches the end of this segment, the value will be held until the key-up signal is generated. If the

key-up signal occurs before this point is reached, the envelope generator will skip to the segment after this one. The integer may be **1**, **2**, or **3**, corresponding to one of the three sustain/release points.

releasepoint <*integer*> *optional, globally once-only*

This sets the specified key-up point to release. This means that the envelope generator will still skip ahead to the segment after this one if a key-up is received, but it will not sustain if it reaches this this point before receiving the key-up signal; instead it will continue as if there were no point. The integer may be **1**, **2**, or **3**, corresponding to one of the three sustain/release points.

sustainpointnoskip <*integer*> *optional, globally once-only*

This sets the specified key-up point to sustain with no skip. This is like sustain above, except that the envelope generator will not skip to this point if the key-up signal is received before it reaches this point. The integer may be **1**, **2**, or **3**, corresponding to one of the three sustain/release points.

releasepointnoskip <*integer*> *optional, globally once-only*

This sets the specified key-up to release with no skip. This is equivalent to not having any key-up point specified, and is included for completeness.

ampaccent1 <*number*> *optional (default is 0)*
ampaccent2 <*number*> *optional (default is 0)*
ampaccent3 <*number*> *optional (default is 0)*
ampaccent4 <*number*> *optional (default is 0)*

Specify the value to scale the specified accent by when adjusting the target level of this segment. Accents are specified as powers of two, so the target level is multiplied by $2^{-\text{ accent } * \text{ ampaccent}}$.

ampfreq slope <*number*> **center** <*number*> *optional*

Specify the control values for envelope scaling by pitch. The first number is the rolloff per octave. A value of zero causes pitch to be ignored; -1 doubles value with each octave; and 1 halves value with each octave. The second number is the frequency at which no change ever occurs. The target level is multiplied by $2^{-\text{ lg(freq / refpnt) } * \text{ ampfreq}}$.

rateaccent1 <*number*> *optional (default is 0)*
rateaccent2 <*number*> *optional (default is 0)*
rateaccent3 <*number*> *optional (default is 0)*
rateaccent4 <*number*> *optional (default is 0)*

Specify the value to scale the specified accent by when adjusting the duration of this segment. Accents are specified as powers of two, so the target rate is multiplied by $2^{-\text{ accent } * \text{ rateaccent}}$.

ratefreq slope <*number*> **center** <*number*> *optional*

Specify the control values for envelope scaling by pitch. The first number is the rolloff per octave. A value of zero causes pitch to be ignored; -1 doubles value with each octave; and 1 halves value with each octave. The second number is the frequency at which no change ever occurs. The target rate is multiplied by $2^{-\text{ lg(freq / refpnt) } * \text{ ampfreq}}$.

exponential or **linear** *optional (default is linear)*

Specify the curve shape for this segment. Linear means the amplitude arithmetically changes as time passes. Exponential means the logarithm of the amplitude arithmetically changes as time passes.

origin ; *optional*

The origin specifies the stage of the envelope that should be synchronized with the origin of any samples, other envelopes, and the start of the note.

Sample List Specification:

The sample list is used for selecting different samples or wave tables based on the pitch of the note. It consists of a list of pairs of sample or wave table names and ceiling pitch values:

<samplename> <max-frequency> **;** *required, repeatable*

For instance, in the following specification:

“Sample1” 261;
“Sample2” 440;
“Sample3” 880;

The sample “Sample1” would be used for all notes below middle C (261.6 Hertz), “Sample2” would be used for all notes between middle C and concert A, and “Sample3” would be used for all notes between concert A and one octave above concert A. Notes beyond the last frequency specified will not be heard at all.

Track Effect Specification:

A track effect is an effect that is applied to the combined output of all active notes on the track.

delayline (*<delaylinelems>* **)**

This effect implements a general purpose delay line. Multiple taps can be specified which add scaled values from within the delay line to the current input signal to produce a new output signal. Taps can also be applied to other elements within the delay line. The following specifiers are allowed within the *<delaylinelems>* list:

maxdelaytime *<time>* *optional (default is 0)*

This sets the maximum delay time. Any delay times that are larger than this will be set to this value.

tap *<tapchannel> <fromtime>* **to** *<tapchannel> <totime>* **scale** *<amplification> <tapattributes>*
optional, repeatable

This specifies a single tap in the delay line. A value is taken from the location specified in *<fromtime>*, multiplied by *<amplification>*, and added to the location specified by *<totime>*. The *<tapchannel>* is one of **left**, **right**, or **mono**, and specifies which channel of the delay line should be used. If **mono** is specified as the source, the average of the left and right channels is used. If **mono** is specified as the target, then the scaled value is added to both channels. The following list describes the allowed tap attributes:

sourceaccent1 *<scaling>*	*optional (default is 0)*
sourceaccent2 *<scaling>*	*optional (default is 0)*
sourceaccent3 *<scaling>*	*optional (default is 0)*
sourceaccent4 *<scaling>*	*optional (default is 0)*
targetaccent1 *<scaling>*	*optional (default is 0)*
targetaccent2 *<scaling>*	*optional (default is 0)*
targetaccent3 *<scaling>*	*optional (default is 0)*
targetaccent4 *<scaling>*	*optional (default is 0)*
scaleaccent1 *<scaling>*	*optional (default is 0)*
scaleaccent2 *<scaling>*	*optional (default is 0)*
scaleaccent3 *<scaling>*	*optional (default is 0)*
scaleaccent4 *<scaling>*	*optional (default is 0)*

These attributes specify the effect of the effect accent parameters on various controls. The actual value used is determined by the formula *<nominalvalue>* + *<scaling>* * *<accent>*. Note that the effect accent parameters are *not* the same as the note accent parameters.

movingaveragelowpass *optional*

If specified, a moving average filter will be applied to the tap signal before it is added to the target.

nlproc wavetable *<identifier>* **inputscaling** *<number>* **outputscaling** *<number>* **index** *<number>* *<nlattributes>*

This effect implements a nonlinear processor which uses the input signal as a phase input to the specified wave table. The left edge of the wave table corresponds to a signal value of -1, and the right edge of the wave table corresponds to a signal value of 1. If these values are exceeded, wraparound will occur (the analyzer effect can be used to determine scaling values that prevent wraparound). The input scaling and output scaling can be used to adjust the signal to be within range. The index field specifies the wave table index. The following attributes can be used to adjust the parameters:

inputaccent1 *<number>*	*optional (default is 0)*
inputaccent2 *<number>*	*optional (default is 0)*
inputaccent3 *<number>*	*optional (default is 0)*
inputaccent4 *<number>*	*optional (default is 0)*
outputaccent1 *<number>*	*optional (default is 0)*
outputaccent2 *<number>*	*optional (default is 0)*
outputaccent3 *<number>*	*optional (default is 0)*
outputaccent4 *<number>*	*optional (default is 0)*
indexaccent1 *<number>*	*optional (default is 0)*
indexaccent2 *<number>*	*optional (default is 0)*
indexaccent3 *<number>*	*optional (default is 0)*
indexaccent4 *<number>*	*optional (default is 0)*

These attributes specify the effect of the effect accent parameters on the input scaling, output scaling, and wave table index controls. The actual value is calculated from the formula *<nominalvalue>* + *<scaling>* * *<accent>*.

filter (*<filterelems>* **)**

This effect implements an array of parallel filters. Each filter in the array is applied to the input signal, and the output of each filter is summed to create the output signal. Cascaded filters can be created by specifying multiple filter effects in sequence. Each element in the *<filterelems>* list has the following form:

filter *<filtertype>* **freq** *<number>* [**bandwidth** *<number>*] *<scaling>* *<filterchannel>* *<filterattributes>* **;**
optional, repeatable

The filter type is one of the following:

null:	Filter that passes all frequencies
lowpass:	Generic first order IIR low pass filter
highpass:	Generic first order IIR high pass filter
reson:	Generic second order IIR all pole resonant band pass filter
zero:	Generic seconrd order FIR all zero filter
butterworthlowpass:	Butterworth low pass filter[1]
butterworthbandpass:	Butterworth band pass filter
butterworthhighpass:	Butterworth high pass filter
butterworthbandreject:	Butterworth band reject filter

For low and high pass filters, the **freq** specifies the cutoff frequency. For band pass/reject filters, it specifies the center frequency.

The bandwidth specifies the width of the pass/reject band. This field is omitted for low and high pass filters.

The scaling field specifies the amplitude scaling control for the filter. Most filters have only one scaling mode, called **defaultscaling**. The reson filter has scaling modes **unitymidbandgain** and **unitynoisegain** in addition to defaultscaling. The zero filter has **unityzerohertzgain** in addition to defaultscaling.

[1]Filter algorithms were obtained from Dodge, Charles and Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*, Schirmer Books, New York, 1985, pages 184-190.

The filter channel specifies which channel the filter should be applied to. It is one of **left**, **right**, or **mono**. For left or right, the filter is applied to the specified channel, and silence is passed to the other channel. For mono, the filter is applied separately to each channel with identical control parameters.

The filter attributes field is one of the following:

outputscaling <*scaling*> *optional (default is 1)*

This specifies a scaling factor to multiply the output signal of this filter element by.

freqaccent1 <*scaling*> *optional (default is 0)*
freqaccent2 <*scaling*> *optional (default is 0)*
freqaccent3 <*scaling*> *optional (default is 0)*
freqaccent4 <*scaling*> *optional (default is 0)*
bandwidthaccent1 <*scaling*> *optional (default is 0)*
bandwidthaccent2 <*scaling*> *optional (default is 0)*
bandwidthaccent3 <*scaling*> *optional (default is 0)*
bandwidthaccent4 <*scaling*> *optional (default is 0)*
outputaccent1 <*scaling*> *optional (default is 0)*
outputaccent2 <*scaling*> *optional (default is 0)*
outputaccent3 <*scaling*> *optional (default is 0)*
outputaccent4 <*scaling*> *optional (default is 0)*

These attributes specify the effect of the effect accent parameters on various controls. The actual value used is determined by the formula <*nominalvalue*> + <*scaling*> * <*accent*>.

analyzer <*string*>

The analyzer does not change the signal flowing through it, but instead measures the minimum and maximum amplitude of the signal over the course of playback, and prints this information to a log window when playback is finished. This is useful for calibrating scaling parameters for nonlinear processors. The <*string*> is printed to the log window with the statistics in order to mark where the statistics come from.

General Menu Items

There are several menu items which are almost always available:

File, New: Create a new, empty document.

File, Open: Open an existing document.

File, Close: Close the window that is currently on top.

File, Save: Save changes to the current document.

File, Save As...: Save the current document under a new name. The file containing the original document will not be modified by this command.

File, Set Tab Size...: Adjust the number of spaces per tab for all text fields in the document.

File, Play...: Adjust control parameters and play the score.

File, Play AIFF File: Play an AIFF file produced by *Out Of Phase* or some other application.

Objects, Calculator: Open a calculator window. The calculator will evaluate program expressions typed in and is useful for experimenting with the programming language and testing new functions.

Build, Remove Objects For All: Remove all computed and compiled objects, including compiled function modules and instruments, and algorithmically generated wave tables and samples. This option can be used if the program fails to use the most recent version of some object, or if the functions used to build some object have been changed.

Build, Build All Objects: Compiles any objects which have not yet been compiled or which were modified after previously being compiled. The program does this automatically when needed, so this option is rarely needed.

Windows: This menu contains the names of all currently opened windows and lets you bring any of them to the top.

Main Document Window

The main document window provides access to all of the components of a song by means of scrolling lists of named objects. It also supplies a text field for specifying title, author, and copyright information.

Figure 1: Main Document Window

Objects are selected by clicking on the name. The following menu items can be chosen to perform some action on the currently selected object:

Objects, Edit Object: Open the appropriate editor for the selected object. This operation can also be done by double-clicking on the object’s name.

Objects, Delete Object: Delete the object from the document.

Objects, Copy Object: Copy an object to the clipboard.

Objects, Paste Object: Paste an object from the clipboard into the document.

An object can be duplicated by selecting the object’s name, choosing **Copy Object**, and then choosing **Paste Object**.

An object can be copied from one document to another by selecting the object’s name, choosing **Copy Object**, switching to another document, and choosing **Paste Object**.

Sample Editor

Out Of Phase includes a sample editor which, although not sophisticated, is flexible enough to do basic editing.

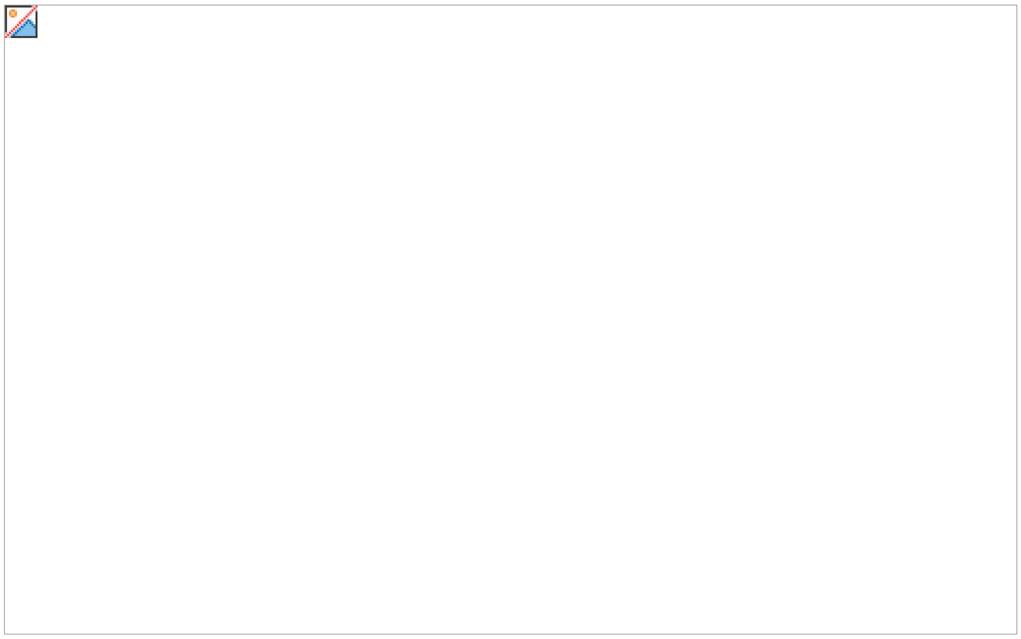


Figure 2: Sample Editor

Sample Name: The name used to refer to the sample in instruments. This name should be unique among sample and algorithmic sample objects.

Sampling Rate: The number of frames per second at which the sample was recorded.

Natural Freq: The natural frequency (pitch) of the sampled sound. This value is used to calibrate note positions on the staff with resampling rates during playback.

Origin: The position within the sample that is synchronized with note key-down and the origin points of all envelopes.

Loop Start: The position within the sample at which a loop starts. Three separate loops can be defined for a sample. The menu items **Sample, Edit Loop 1**, **Sample, Edit Loop 2**, and **Sample, Edit Loop 3** can be used to change which loop is being displayed. The names next to the edit boxes will change accordingly.[2]

Loop End: The position within the sample at which the loop ends. Playback jumps back to the beginning of the sample. If this value is the same as **Loop Start**, then there is no loop.

Origin, **Loop Start**, and **Loop End** strips: These strips graphically display the origin and loop positions. The points can be changed by clicking in the strip.

Sample Edit Field: The sample edit field displays the sample data. Regions can be selected by clicking and dragging in the field.

Speaker Buttons: Select whether the sample is stereo or mono.

[2]All three sets of loop point edit boxes should really be displayed at once. Unfortunately, the window is already over-stuffed with edit boxes and I couldn’t figure out any better way of cramming all three loop point edit boxes into it.

8-Bit, **16-Bit**: Select whether the sample is 8-bit or 16-bit.

Select Start: Start point of the selected section of the sample.

Select End: End point of the selected section of sample. This number is actually the index of the sample frame immediately after the last selected frame.

Mouse Location: Position in the sample currently under the mouse pointer. This field can not be edited.

Scaling Factor: The expansion/compression ratio for displaying the sample. The **Zoom In** button can be used to divide this in half, while the **Zoom Out** button can be used to double it.

Triangle (test play): Test the sample by playing it through the speaker at the recorded sampling rate. The sample will repeat the currently selected loop until the mouse button is released.

The final edit field in the sample editor is the formula field. This field allows arbitrary expressions to be entered in order to allow transformations to be applied to the sample. It is intended that function code will reside in **Function Modules** and only calls will be made; hence the small edit area. In the above example, a function call is made to maximize a quiet sample.

Any modifications apply to the entire sample regardless of any selection; the modification algorithm used must deal with the selected area if that is desired.

The following variables are defined:

loopstart: an integer containing the start point of the current loop
loopend: an integer containing the end point of the current loop
origin: an integer containing the origin of the sample
samplingrate: an integer containing the sampling rate
selectstart: an integer containing the start of the current selection
selectend: an integer containing the end of the current selection
naturalfrequency: a double precision floating point value containing the natural recorded pitch

For mono samples only:
data: a fixedarray containing the sample data

For stereo samples only:
leftdata: a fixedarray containing data for the left channel
rightdata: a fixedarray containing data for the right channel

Only changes made to the **data**, **leftdata**, and **rightdata** arrays are remembered. Change to other variables are ignored.

Code entered in the formula box is executed by choosing the menu item **Objects, Evaluate**. The most recent execution can be undone.

Algorithmic Sample Editor

An algorithmic sample is a sample that is generated computationally, rather than a stored array of data. The algorithmic sample editor provides a way of constructing such samples.

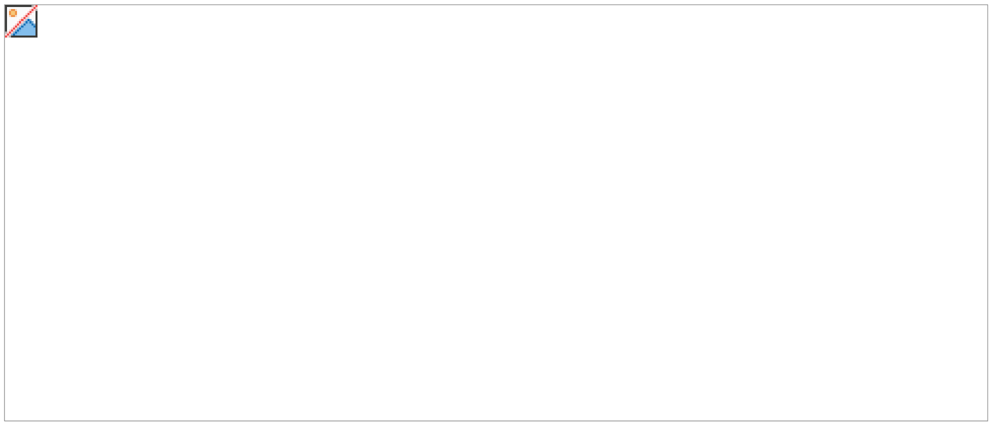


Figure 3: Algorithmic Sample Editor

The edit fields in the algorithmic sample editor have the same function as the corresponding fields in the stored sample editor.

For computation, the following variables are defined:

loopstart1: an integer containing the start point of the first loop
loopend1: an integer containing the end point of the first loop
loopstart2: an integer containing the start point of the second loop
loopend2: an integer containing the end point of the second loop
loopstart3: an integer containing the start point of the third loop
loopend3: an integer containing the end point of the third loop
origin: an integer containing the origin of the sample
samplingrate: an integer containing the sampling rate
naturalfrequency: a double precision floating point value containing the natural recorded pitch

For mono samples only:
data: a fixedarray containing the sample data

For stereo samples only:
leftdata: a fixedarray containing data for the left channel
rightdata: a fixedarray containing data for the right channel

The algorithmic sample can be evaluated and then converted into a sample for testing by choosing the menu item **Build, Open Algorithmic Sample As New Sample**. The resulting copy can be deleted since the program will rebuild the algorithmic sample as needed.

Wave Table Editor

The wave table editor allows stored wave tables to be displayed and minimally edited. Full editing is not currently supported since stored wave tables are not intended to be used in practice; instead algorithmic wave tables will usually be used.[3]

Figure 4: Wave Table Editor

Name: The name by which this wave table is referred to. The name should be unique among wave tables and algorithmic wave tables.

Frames: The number of frames in a wave period. Since this number must be an integral power of two, clicking in this box will present a dialog of choices. Changing the number of frames will cause the program to resample the waveforms.

Tables: The number of wave periods in the table. Changing this number will cause the program to interpolate between periods in order to generate a new set.

8-Bit, **16-Bit**: The number of bits used to store the points in the wave table.

Test Attack: The number of seconds of table index attack when testing the wave table.

Test Decay: The number of seconds of table index decay when testing the wave table.

Test Freq: The pitch (in Hertz) at which the table should be played when testing.

Test Smpl Rate: The sampling rate to use when testing the wave table.

Test: This button starts a test of the wave table. A test is performed by playing each wave period in succession for some amount of time, and then playing each period in reverse succession.

Waveform Display: The scroll bar can be used to inspect each wave period in the table.

[3]Stored wave tables would be useful for wave tables derived from acoustic instruments. Unfortunately, automatically converting an acoustic instrument sample into a wave table is very difficult.

Formulas for augmenting the wave table can be specified in the edit field. This area functions much like the formula box in the sample editor, except that the variables are different.

frames: An integer containing the number of frames per period
tables: An integer containing the number of periods in the table
data: An array of data containing the wave periods. There are **frames** * **tables** entries in the data array. The array contains the periods packed, in ascending order of index.

Algorithmic Wave Table Editor

The algorithmic wave table editor allows programs to be specified which build the waveform.

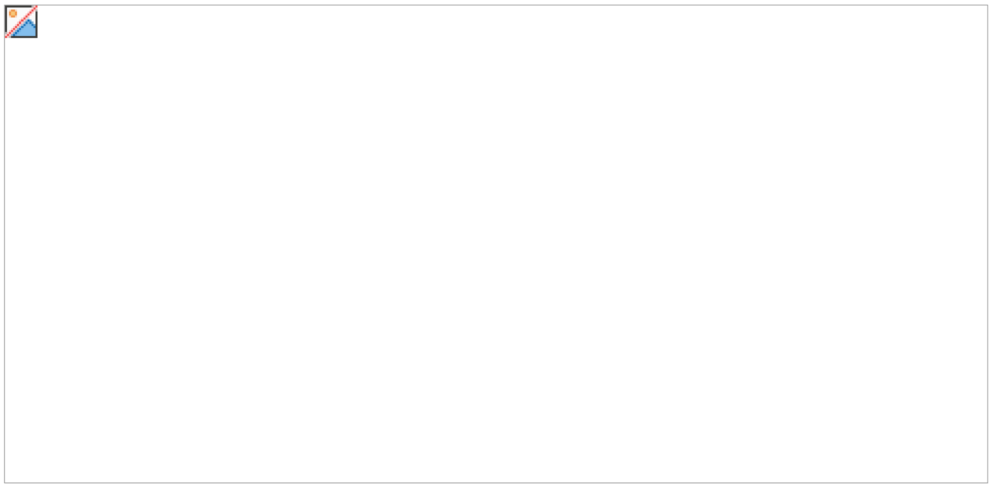


Figure 5: Algorithmic Wave Table Editor

The parameters for the algorithmic wave table editor are similar in function to those of the stored wave table editor.

The algorithmic wave table can be evaluated and then converted into a stored wave table for testing by choosing the menu item **Build, Open Algorithmic Wave Table As New Wave Table**. The resulting copy can be deleted since the program will rebuild the algorithmic wave table as needed.

Function Module Editor

The function module editor provides a way of storing important functions in a place where all other editors can find them.

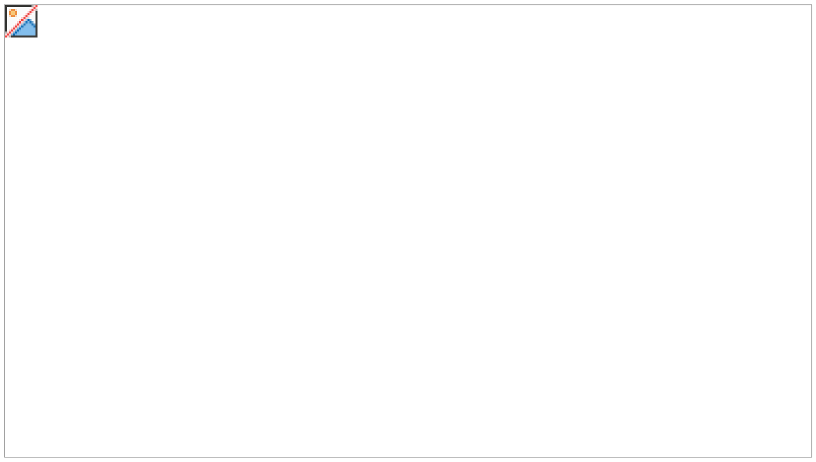


Figure 6: Function Module Editor

The function name field is ignored by the program.

Any number of functions can be specified in a function module. The following menu items are useful for editing and debugging functions:

Build, Compile Function Module: This will attempt to compile the function module. If there is a syntax error, the line containing the error will be highlighted and an explanation of the error given.[4]

Disassemble Function Module: Functions are compiled internally into an interpreted bytecode. This menu item will compile the functions and then disassemble the bytecode. This is generally only useful for programmers who are looking for bugs in the compiler.

[4]Actually, a line *near* the error will be highlighted. The error detection mechanism is not always accurate about which line contains the error. For instance, if the error is near the end of a line, the next non-empty line may be highlighted instead.

Instrument Editor

The instrument editor allows instrument specifications to be written and debugged.

Figure 7: Instrument Editor

The instrument name field specifies the string by which tracks will refer to instruments. This field should be unique among the instruments.

An instrument can be built by choosing the menu item **Build, Build Instrument**. If there is a syntax error, the line containing the error will be highlighted and an explanatory dialog box displayed.[5]

[5]The caveat given for syntax errors in function modules also applies here.

Track Editor

The track editor is the heart of the system. It allows scores to be programmed in pseudo-standard notation. Each track object represents the score for one instrument.

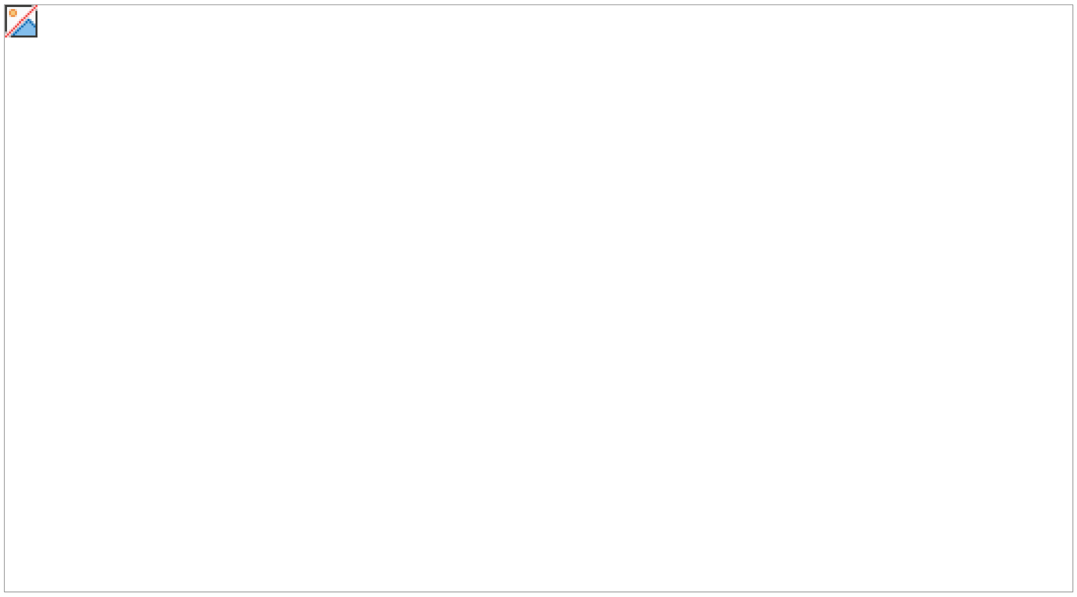


Figure 8: Track Editor

The buttons along the top of the window select what editing mode the window should be in. The arrow represents selection mode which allows portions of the score to be selected for copy and paste. The icon to the right, with a small box, is for inserting control commands into the score. The other buttons control the type of note that will be added to the score.

Chords are spread out horizontally to allow selection of individual notes in the chord. The space between notes in a chord is less than the space between chords. The duration of the chord is the duration of the shortest note in the chord.

Measure bars are automatically displayed. A black bar indicates that the previous chord ended exactly at the end of the previous measure. A grey bar indicates that the previous chord extended into the next measure and the bar could not be placed precisely at the measure boundary.

Keyboard Shortcuts

There are several keyboard shortcuts for selecting entry modes and note types. These keyboard shortcuts are used *without* the command key; the command key continues to serve as a shortcut for menu choices.

Arrow Keys: Scroll the score a little in some direction.
‘**/**’: Reset the note entry mode to a quarter note with no special forms selected.
‘**a**’: Switch to selection mode (same as clicking on the arrow icon).
‘**z**’: Switch to note insertion mode using the most recent note palette selections.
‘**c**’: Switch to command insertion mode (same as clicking box icon).
‘**x**’: Select 64th note.
‘**t**’: Select 32nd note.
‘**s**’: Select 16th note.
‘**e**’: Select 8th note.
‘**q**’: Select quarter note.

‘**h**’: Select half note.
‘**w**’: Select whole note.
‘**d**’: Select double note.
‘**f**’: Select quad note.
‘**=**’, ‘**+**’: Select sharp.
‘**-**’, ‘**_**’: Select flat.
‘**0**’: Select natural.
‘**n**’: Select note.
‘**r**’: Select rest.
‘**,**’: Select no dot.
‘**.**’: Select dot.
‘**1**’: Select no division.
‘**3**’: Select triplet division.
‘**5**’: Select quintuplet division.
‘**7**’: Select septuple division.
‘**>**’: Transpose selection up one step.
‘**<**‘: Transpose selection down one step.

Adding Notes

To add notes to the score, select the note buttons to determine the appropriate type of note you with to insert. Then position the mouse at the point where the note should be inserted. If the mouse is positioned over an existing note, the new note will be added to the existing note as a chord; the cursor will look like a cross hair without arrows on the ends. If the mouse is positioned between chords, the new note will be inserted between the chords; the cursor will look like a cross hair with arrows on the ends. The heavy horizontal bar indicates what staff line the note will be added at. To add the note, click the mouse button.

Selecting and Removing Notes

Single notes can be selected by command-clicking on the note. A grey selection rectangle will surround the column containing the note. To delete the note, press the delete key.

A whole chord or range of chords can be selected by clicking on the arrow icon and dragging the mouse from the beginning of the first chord to the end of the last chord. The selection will be surrounded by black rectangle. A selection can be deleted or copied.

Adding Commands

To insert a command, choose the command insertion icon and click at the point where the command should be inserted. Choose the command from the dialog box. After the command has been inserted, hit return to edit the command’s parameters.

Commands control global parameters, allowing the parameters specified in the track defaults dialog box to be changed during playback (see *Editing Track Attributes* below). There are a variety of commands available. Most have several variants:

Restore: Restore the parameter to the value it had when playback started.

Set: Specify a new value for the parameter.

Adjust: Specify some adjustment that will be used to change the parameter. In most cases, the adjustment value is added to the current parameter value.

Sweep Absolute: Sweep the value of the parameter from the current value to a specified target value, over a specified number of beats (quarter notes).

Sweep Relative: Sweep the value of the parameter from the current value to a specified adjustment of the current value, over a specified number of beats.

Additionally, some parameters have modes that can be adjusted, such as the release point origin or frequency displacement mode.

The following parameter change classes are available:

Tempo: Change the tempo. The tempo for the entire score is changed, not just the track containing the command.

Stereo Position: Change the default stereo position/pan value for the track.

Volume: Change the default maximum volume of subsequent notes played on the track. The **adjust** variant multiplies the current volume by the specified value.

Release Point 1, 2: Adjust the default locations of the release points.

Accent 1, 2, 3, 4: Adjust the default values of accent parameters.

Pitch Displacement Depth: Adjust the default maximum pitch LFO displacement depth.

Pitch Displacement Rate: Adjust the default maximum pitch LFO rate.

Pitch Displacement Start: Adjust the default start point of the pitch LFO generator.

Hurry Up: Adjust the default envelope rate acceleration factor. The **adjust** variant multiplies the current hurry up factor by the specified value.

Detune: Adjust the specified pitch detuning value.

Early/Late Adjust: Adjust the default amount of beat lead or lag.

Duration Adjust: Adjust the default duration expansion/compression factor.

Set Meter: Change the number of beats displayed per measure. This does not affect playback.

Set Measure: Set the number of the next measure. This does not affect playback.

Set Transpose: Set a transpose value for subsequent notes on the track. The value is in half-steps; negative values transpose down and positive values transpose up.

Adjust Transpose: Add a value to the current transpose parameter.

Marker: Add a text comment to the score. This does not affect playback.

Setting and Clearing Ties

A tie or slur can be set by first selecting the source note by command clicking on it, then selecting the target note by control clicking on it.

A tie or slur can be cancelled by first selecting the source note by command clicking on it, then control clicking on or before the same note.

Editing Track Attributes

Track attributes can be edited by choosing the **Objects, Edit Track Attributes** menu item. This will display a dialog of all the default track parameters.

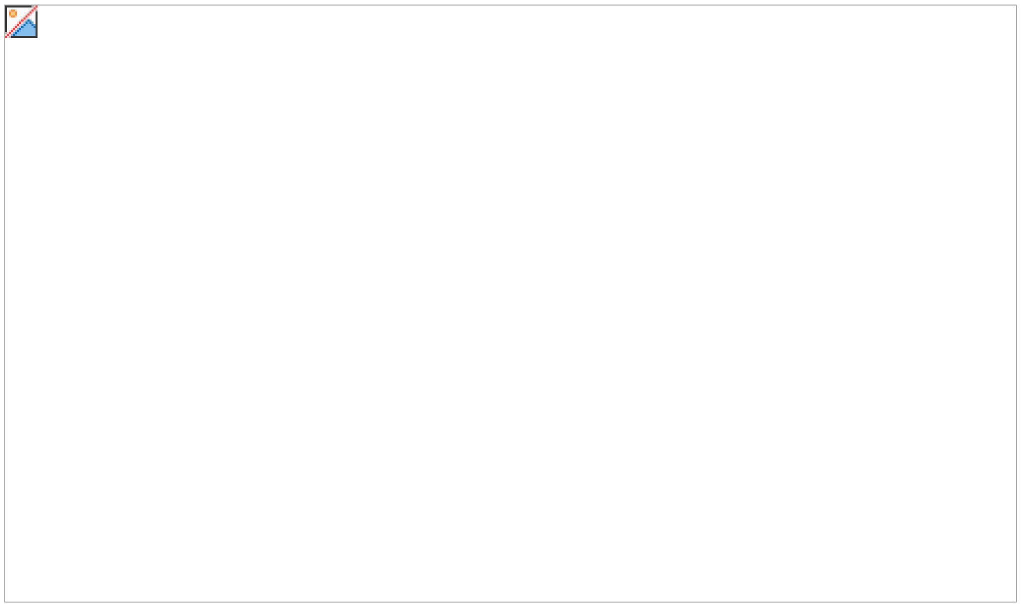


Figure 9: Track Attributes Dialog

The parameters in the track attributes determine the default controls on synthesis. These can be adjusted for each note by editing the note attributes. These are the parameters affected by the commands specified above. These parameters are combined with per-note parameters to determine how each note will be played. Their effects are described below.

Editing Note Attributes

The attributes for a specific note can be edited by command clicking on the note and then hitting return. This will present the note attribute dialog.

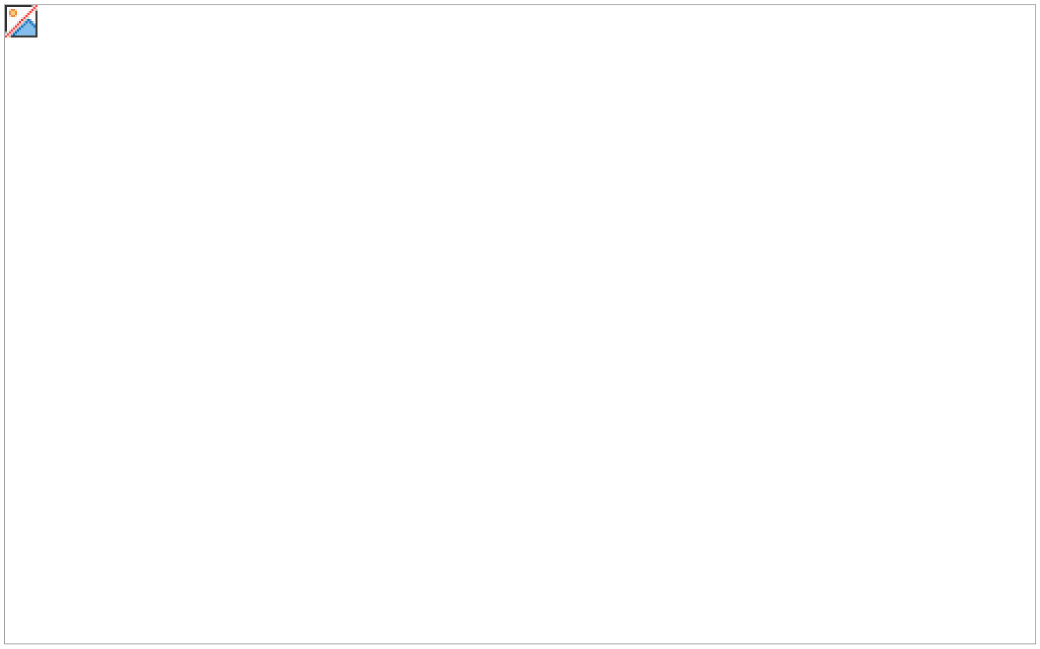


Figure 10: Note Attribute Dialog

These parameters augment the parameters specified in the track attributes dialog. Here is a detailed description of what these parameters do:

Hit Time: This value adjusts when the note actually begins. Negative values make the note lead the beat slightly, while positive values make the note lag the beat slightly. A magnitude of 1 corresponds to a displacement as long as the note's duration. This value is added to the track default value to determine the actual displacement.

Loudness: This value adjusts the maximum loudness of the note. It is multiplied with the instrument's overall loudness and the track default loudness to determine the actual amplitude scaling factor.

Release Point 1, 2: This specifies the position of the release point. If **start** or **end** is selected, then only this value is used in determining the release point, otherwise the value is added to the track default value, and the track default **start/end** setting determines the origin. An origin of **start** means the value is the fraction of the note's duration after the note begins that the release point will occur; larger values mean the release point is closer to the end of the note. And origin of **end** means the value is the fraction of the note's duration *before* the note end that the release point occurs; note that in this case larger values mean closer to the *beginning* of the note.

Accent 1, 2, 3, 4: These specify the accent values that are passed into the envelope generator for adjusting envelope segments with **accent** attributes. Each accent value is added to the corresponding track default value.

Pitch Disp Rate: This specifies the maximum pitch LFO rate, in cycles per second. It is multiplied by the track default pitch displacement rate.

Pitch Disp Depth: This specifies the maximum deflection of the pitch LFO. This value is multiplied by the track pitch displacement depth. The resulting value is used to scale the levels of the pitch LFOs.

Pitch Disp Start: This specifies the start point to which the origin of the pitch LFO is synchronized. The operation of this parameter is identical to the operation of the release point parameters.

Hurry-Up: This specifies an acceleration parameter for the envelope generators. The value is multiplied by the track default value and then used to multiplicatively scale each envelope segment's duration. Values larger than 1 increase the duration, and values smaller than 1 decrease the duration.

Stereo Pos: This specifies the stereo position of the note, in a range from -1 to 1, where -1 is hard left, 0 is center, and 1 is hard right.

Detune: This specifies pitch detuning for the note. If **hertz** or **halfsteps** is selected, then the value is added to the track default value and interpreted as either a Hertz frequency change or halfstep change to be applied to the note's pith. If **default** is selected, then the value is *multiplied* by the track default value, and the track default setting is used to determine how to apply the value to the note's pitch.

Duration: This is a symbolic entry field for the note's duration.

Duration Adjust: This adjusts the note's duration during playback. If **add** is selected, then the value is the number of quarter note durations to add to the note's nominal duration. If **mult** is selected, then the value is multiplied by the note's duration. If **default** is selected, then the track default add/mult setting is used. The track default duration adjust is then used to adjust the note's duration.

Real Pitch: This is a symbolic entry field for the note's pitch.

Effective Pitch: This is a symbolic entry field for a pitch value to be used for selecting samples or wave tables for multisampled instruments. If it is **default**, then the real pitch is used for selecting samples or wave tables.

Portamento: This specifies the number of quarter note durations that a portamento slide should last. A value of 0 disables portamento slide. The **hertz** or **halfsteps** setting determines how the slide will occur.

Note Is Actually a Rest: This check box allows a note to be converted to a rest or vice versa.

Retrigger Envelopes on Tie: If this check box is set, then all envelope generators will be restarted from their origins when a tie-follower note begins. Under all conditions, the envelope parameters are recalibrated using the new note's accent parameters.

Release 3 From Start Instead: Normally, the third release point occurs at the end of a note's duration period. This check box can cause it to occur at the beginning of a note's duration period instead (i.e. at key-down).

Viewing Tracks in the Background

While laying out a track, it is sometimes helpful to be able to see another track in the background for reference. The **Tracks** menu allows other tracks to be displayed along with the current track.

Sample Import and Export

The sample editor built into *Out Of Phase* is not powerful enough to do extensive sample editing, and the program lacks the ability to record sounds. To allow other programs to be used for these purposes, *Out Of Phase* can import and export samples in a couple of popular file formats. The following menu items under the Sample menu allow this:

Import WAV Sample: Import a sample in the Microsoft RIFF format (with a filename extension ".WAV") and open a new editor. These files usually come from MS-Windows and MS-DOS programs.

Import AIFF Sample: Import a sample in the Apple AIFF format and open a new editor. These files usually come from Macintosh programs, such as SoundEdit. The AIFF-C format extension is recognized, but compression is not supported.

Import Raw Sample: If you know the structure of the file containing the sample you wish to import, this command provides some reasonably flexible file structure specifications. The following dialog box is presented:

Figure 11: Raw Sample Import Dialog

Initial Skip: The number of bytes in the file to skip before starting to read samples.

Skip Between Frames: The number of bytes to skip between successive sample frames. A sample frame is a set of sample points that define the signal at a moment in time. For stereo, there are two sample points in each frame. Normally, frames are packed, so this value would be zero.

Number Of Bits: The number of bits that define a sample point. If a sample point is one byte, then use **8-bit**; if it is two bytes, then use **16-bit**.

Sample Sign: The sign encoding method for sample values. **Signed** is 2's complement binary. **Unsigned** means sample points are always positive values. **Sign Bit** means the upper bit of each sample point is clear for positive values and set for negative values; the rest of the number is treated as unsigned.

Endianness: The byte ordering for 16-bit sample points. Files from the Macintosh and UNIX world usually use big endian ordering, whereas files from the MS-DOS and MS-Windows world usually use little endian ordering.

Number Of Channels: The number of channels the file has. For mono, each sample frame contains one sample point. For stereo, each sample frame contains two sample points, with the left point preceding the right point.

If a sample editing window is open, then the following additional menu items are available:

Export WAV Sample: Export the sample in Microsoft RIFF format.

Export AIFF Sample: Export the sample in Apple AIFF format.

Export Raw Sample: Export the sample in a simple user-definable form. This is often useful if you want to write a small program to process sound files but you don’t want to worry about handling standard file formats. Choosing this item brings up the following dialog box:

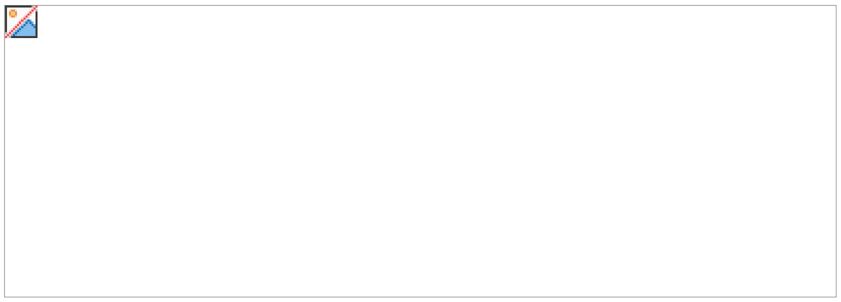


Figure 12: Raw Sample Export Dialog

The parameters have the same function as the corresponding parameters specified in the import dialog box. The number of bits and channels is specified in the sample editor. The file written is packed sample frames with no alignment or padding.

Playing the Song

The song can be played by selecting **File, Play...**. This will present a dialog box of control parameters.

Figure 13: Play Dialog

The scrolling list on the left allows tracks to be enabled or disabled from the output. Highlighted tracks are included in the output.

The following parameters control the quality of the output:

Sampling Rate: This determines the number of frames per second in the output. CD quality is 44100. A highest quality sampling rate for older Macintoshes is 22254. In general, a higher sampling rate is better, but requires more computation time, and more disk space if a file is generated.

Envelope Rate: This determines the number of times per second that values controlled by envelopes will be adjusted, as well as when notes are started. Higher values are better but require significantly more computation. Values as low as 100 can produce acceptable results. In general, a value of 1000 produces decent results for final playback, although even larger values may be desirable.

Beats Per Minute: This is the default tempo. It may be adjusted during playback by using the tempo command.

Inverse Volume: The inverse volume controls the loudness of the output. Larger values result in quieter output. The value should be set to the maximum number of full volume sounds generated at the same time. In general, experimentation is required to find the best value. Clipping occurs if it is set too low; meaning that some sample frames exceed the range of the channel and are constrained at the maximum or minimum channel values. If it is set very low, clamping may occur, meaning that a volume scaling parameter exceeded the internal range of the resampling function and was constrained at the maximum point, resulting in distortions of instrument volumes relative to each other. If either occurs, a dialog box is presented at the end of synthesis indicating how severely it occurred and the minimum inverse volume required to eliminate the problem. The clipping warning can be disabled by deselecting the clip warning check box.

Scanning Gap: The synthesizer has the ability to start notes before the score indicates the note should begin, to correct for instruments with slow attacks and to provide the ability to lead the beat. In order to implement this, a delay between note scheduling and note playback is required. The scanning gap specifies this delay in seconds and should be at least as long as the longest pre-origin time used in the score.

Buffer Seconds: This specifies the number of seconds of buffering to use for audio device playback.

n-Bit Output: This specifies the quality of output. For audio playback, only 8 and 16 bits are significant; higher settings are treated as 16 bits. For playback to disk, all settings are recognized. All computation is performed internally at a precision of 24 bits.

8-bit: Generate 8-bit output.
16-bit: Generate 16-bit output. This is the same as a CD player.
24-bit: Generate 24-bit output. This is intended to be used by people who will do further mixing of the sound, or who have equipment that supports more than 16 bits.
32-bit: Output the raw 32-bit data from synthesis. This mode does not provide significantly more precision than 24 bits but does not perform clipping. This is intended for people who need the maximum precision and want to do their own volume scaling.

Mono/Stereo: This selects whether one or two channels will be output. The inverse volume must be set lower for stereo than for mono, since the signal is split between two channels. In general, more severe panning requires less reduction of inverse volume when going from mono to stereo.

Interpolate: This determines how much interpolation will be used. Interpolation improves sound quality but requires more processing time. **Interpolate Over Time** will interpolate samples and wave tables from one sample frame to the next. **Interpolate Across Waves** will also interpolate between adjacent wave periods for indexed wave table synthesis resulting in smoother transitions.

Cancel: Close the dialog box and forget any changes that were made to the parameters.

Done: Close the dialog box and remember any changes that were made to the parameters.

Play To Audio: Play the sound to the audio device in realtime. It is quite likely that the computer will not be able to generate sample frames fast enough to keep up with the audio device, resulting in gaps during playback. The settings can be adjusted to use less processing power (with a corresponding sacrifice in sound quality), or the song can be played to disk.

Play To Disk: Play the sound to an AIFF-C file on disk.

AIFF-C files generated with the **Play To Disk** option can be played back by choosing the **File, Play AIFF File** menu option.

While editing, it is sometimes useful to start playing from some point in the song other than the beginning. To do this, select the desired starting point from the track editor. The menu item **File, Play This Track** will play only this track to the audio device using the parameters set in the play dialog. The menu item **File, Play All Tracks** will play this track along with any other tracks selected in the play dialog. If the menu option is selected while holding the option key down, the sound will be played to disk instead of to the audio device.