Section 4

Humdrum Command Reference

Command Documentation Style

This section of the *Reference Manual* provides detailed information for all of the commands in the Humdrum Toolkit. These commands allow various Humdrum representations (described in Section 2) to be manipulated or transformed in musically useful ways. The commands can be classified into two broad groups: basic tools and specialized tools. *Basic tools* can be applied to any type of Humdrum input. An example of a basic tool is the **pattern** command — which is able to locate user-defined patterns for any type of Humdrum input. *Specialized tools* are more restricted in their operation. For example, the **cents** command changes pitch representations into hundredths of semitones. The tables on the following pages list all the basic and specialized commands included in Release 1.0 of the Humdrum Toolkit.

Note that all Humdrum commands assume that inputs conform to the Humdrum representation syntax. If a command fails to perform in the expected fashion, the user should first ensure that the input is valid by invoking the **humdrum** command. Most commands do not check the syntax of their inputs, so invalid inputs will produce invalid outputs.

Each command is described in a separate manual entry. Reference information is organized according to the standard UNIX documentation style. For users unfamiliar with this style, the following description should prove useful in understanding common conventions and abbreviations.

In general, commands are invoked according to the following conventional order of elements:

command name   *options & arguments*   *input files*   *output file*

The *command name* is the name of the executable program to be invoked; *options* modify the operation of the command in distinctive ways; *arguments* provide special information required by some options; when more than one option is specified, each option may be followed by its own argument; *input files* are the names of pre-existing files of information to be processed; *output file* is the name of some file that will store the result.

The syntax by which a given command is invoked is specified in the **SYNOPSIS** section of the command documentation. **Bold** typeface indicates material to be typed literally by the the user. *Italic* typeface indicates text to be supplied by the user — such as file names, regular expressions, or numerical values. For example, the word *inputfile* should be replaced by the name of the file intended by the user as input. The italic letter *n* refers to an integer number (such as the number '23') to be supplied by the user. The italic *n.n* refers to a real number (such as the number '5.31') to be supplied by the user. Ellipses (...) are used to indicate that the preceding item may be repeated any number of times. Thus

inputfile ...

means that the user can specify one or more files as input. Square brackets [ ] are used to indicate items that are optional and may be omitted.

BASIC TOOLS	
assemble	paste together Humdrum files
census	determine general properties of a Humdrum input
cleave	join tokens from two or more spines into a single spine
context	congeal data records to form a contextual frame
correl	measure the numerical similarity between two spines
encode	interactive Humdrum encoding from MIDI input
extract	select input spines for output
fields	trace changes in spine structure
fill	replace null tokens with previous non-null data token
humdrum	test conformance to Humdrum syntax
humsed	stream editor for Humdrum files
info	calculate information flow
num	number selected records according to user-defined criteria
patt	locate and output user-defined patterns in a Humdrum input
pattern	exhaustively locate user-defined patterns in a Humdrum input
recode	recode numeric tokens in selected Humdrum spines
rend	split tokens in a single spine into two or more spines
rid	eliminate specified record types from the input
scramble	randomize order of either Humdrum data records or data tokens
simil	measure the similarity between two Humdrum spines
strophe	selectively extract strophic data
thru	expand repeats to through-composed form
xdelta	calculate numeric differences for successive tokens within a spine
yank	extract passages from a Humdrum input
ydelta	calculate numeric differences for concurrent spines

Basic Humdrum commands

The dash character (-) is used to designate command options — and should be typed literally when an option is invoked. Options are usually indicated by a single alphabetic letter, such as the **-b** option. In many cases, the option is followed by an *argument* that specifies further information pertaining to the invoked option. For example, the syntax: **-f** *filename* means that the name of a file must be provided as an argument if the **-f** option is specified. Options and their accompanying arguments must be separated by blank space (i.e. one or more spaces and/or tabs). If more than one option is invoked, and none of the invoked options require an argument, then the option-letters may be concatenated together. For example, the **-a** and **-b** options might be invoked as **-ab** (or as **-ba**) — provided neither option requires an argument.

By way of example, consider the syntax for the transposition command, **trans:**

trans -d [±]*n* [**-c** [±]*n*] [*inputfile* ...] [> *outputfile*]

Since the word **trans** is in bold typeface, the user must type it literally in order to invoke the command. The **-d** is an "option"; however, since this option is not surrounded by square brackets it is actually required. The user must type, literally:

```
trans -d
```

SPECIALIZED TOOLS

cbr	calculate the critical band rate corresponding to some frequency
cents	translate pitch-related representations to cents
deg	translate pitch-related representations to relative scale degree
degree	translate pitch-related representations to absolute scale degree
diss	calculate the degree of sensory dissonance for successive spectra
freq	translate pitch-related representations to frequency
hint	determine harmonic intervals between concurrent pitches
iv	determine interval vectors for successive vertical sonorities
humver	display Humdrum toolkit version and copyright information
kern	translate pitch-related representations to kern
key	estimate the key (tonic and mode) of a passage
melac	calculate melodic accent values for successive pitches
metpos	assign metric position indicators to sonorities
mint	determine sequential diatonic interval between successive pitches
nf	determine normal form for successive vertical sonorities
pc	translate numeric semit values to numeric pitch-class
pcset	convert pitch-class to set-theoretic representation
pitch	translate pitch-related representations to American standard pitch notation
proof	check syntax of kern file
regexp	interactive regular-expression tester
reihe	output specified row variant for a given prime row
semits	translate pitch-related representations to semits
solfg	translate pitch-related representations to French solfège notation
spect	assemble total spectral content for individual sonorities
synco	measure degree of metric syncopation
timebase	reformat kern score with constant time-base
tonh	translate pitch-related representations to German pitch notation
trans	transpose pitch representations
urrhythm	characterize the rhythmic prototypes in a passage
vox	determine number of concurrently active voices

Specialized Humdrum commands

Since the letter *n* is in italics, it means that an integer must be specified. The integer is preceded by "±" in square brackets; this means that the number may be preceded by an optional plus or minus sign, such as:

```
trans -d +3
```

or

```
trans -d 6
```

or

HUMDRUM MIDI TOOLS	
encode	interactively encode Humdrum data from MIDI input
midi	convert from `**kern` to Humdrum `**MIDI` format
midreset	reset MIDI controller card
perform	play Humdrum `**MIDI` files
record	record MIDI activity in Humdrum `**MIDI` data format
smf	create *Standard MIDI File* from Humdrum `**MIDI` input
tacet	reset MIDI channels; silence

Humdrum MIDI commands

```
trans -d -12
```

Another option — the **-c** — option is also presented in square brackets. The entire expression is:

[**-c** [±]*n*]

This whole expression may be omitted. However, if the user chooses to use this option, the syntax *within* the square brackets must be followed. The **-c** is in bold typeface, so the user must type it literally if the option is selected. The letter 'c' is followed by blank space which can be followed by an optional plus or minus sign, followed by a required integer number. Syntactically correct invocations of the **trans** command include the following:

```
trans -d 13 -c 2
trans -d +7
trans -d -1 -c -5
trans -d -4 -c +8
```

If desired, the user may then specify one or more input files. Each file must be separated by blank space. Finally, an optional *outputfile* may be specified — however the name of the file must be preceded by the file redirection symbol ('>').

Inputs and Outputs

Most commands support several input and output modes *not explicitly indicated in the command* **SYNOPSIS**. Input to a command may come from three sources. In many cases the input will come from an existing file. Apart from existing files, input may also come from text typed manually at the terminal, or from the output of preceding commands. When input text is entered manually it must be terminated with an end-of-file character (control-D) on a separate line. (On IBM PCs the end-of-file character is control-Z.) When input is received from preceding commands, the output is sent via a UNIX pipe ('|').

The different ways of providing input to a command are illustrated in the following examples. In the first example, the input (if any) is taken from the terminal (keyboard). In the second example, the input is explicitly taken from a file named `input`. In the third example, the input is implicitly taken from a file named `input`. In the fourth example, the input to `command2` comes from the output of `command1`.

```
command
command < input
command input
command1 | command2
```

Outputs produced by commands may similarly be directed to a variety of locations. The default output from most commands is sent to the terminal screen. Alternatively, the output can be sent to another process (i.e. another command) using a pipe (|). Output can also be stored in a file using file redirection operator ('>') or *added* to the end of a (potentially) existing file using the file-append operator ('>>'). In the first example below, the output is sent to the screen. In the second example, the output is sent to the file `outfile`; if the file `outfile` already exist, its contents will be overwritten. In the third example, the output is appended to the end of the file `outfile`; if the file `outfile` does not already exist, it will be created. In the fourth example, the output is sent as input to the command `command2`.

```
command
command > outfile
command >> outfile
command1 | command2
```

When two or more commands have their inputs and outputs linked together using the pipe operator (|), the entire command line is known as a *pipeline*. Pipelines occur frequently in Humdrum applications.

Tee

A special UNIX command known as `tee` can be used to clone a copy of some output, so that two identical output streams are generated. In the first example below, the output is piped to `tee` which writes one copy of the output to the file `outfile` and the second copy appears on the screen. In the second example, the output from `command1` is split: one copy is piped to `command2` for further processing, while an identical copy is stored in the file `outfile1`; if the file `outfile1` already exist, its contents will be overwritten. In the third example, the append option (**-a**) for `tee` has been invoked — meaning that the output from `command` will be added to the end of any existing data in the file `outfile`. If the file `outfile` does not already exist, it will be created.

```
command | tee outfile
command1 | tee outfile1 | command2 > outfile2
command | tee -a outfile
```

The `tee` command is a useful way of recording or diverting some intermediate data in the middle of a pipeline.

Quotation Marks in the Command Line

Commands are recognized and executed by a command-line interpreter, known as a *shell.* Several characters and keywords hold special meanings for the shell. As we have learned, the characters > and | are normally used to mean *file redirection* and *pipe* respectively. In addition, the characters < & ( ) ; ` ' " * ? . # % = $ [~ and \ all have special meanings. For example, the characters * and ? are treated as pattern *metacharacters;* the asterisk means "match all instances" whereas the question-mark means "match any single letter." Hence, the command:

```
command *a?
```

will apply the `command` action to all files in the current directory having the letter 'a' as the second last letter in their filenames.

These special characters can be a nuisance when the user wants them to be treated literally, rather than according to their special meanings. The simplest way to defeat these meanings is by preceding the special character by a backslash character (\). The backslash is refered to as an "escape character" since it releases other characters from their special designations.

An alternative way of directing the shell command interpreter to disregard the meanings of special characters is by using quotation marks. Both single quotes (apostrophes) and double quotes can be used, however, single quotes are *stronger* in their effect. Consider the following commands:

```
echo $a
echo "$a"
echo '$a'
```

The dollar-sign ($) is interpreted by the command interpreter as specifying a shell variable. In the first and second commands, the shell looks for a variable named `$a` and attempts to echo the contents of this variable on the display. Unless `$a` happens to be an active shell variable, only an empty line will be displayed. In the third command, the string `$a` is treated literally, and echoed back to the display. There are circumstances where the double quotes are more useful, but for most casual users, the single quotes provide the best means for disengaging the meanings of special characters.

For a more complete discussion of command-line quoting, refer to any standard UNIX manual.

Help

Nearly all of the Humdrum commands provide a **-h** option that causes a "help screen" to be displayed. Each help screen simply provides a brief statement of the purpose of the command, the types of permissible input formats (if applicable), a brief summary of the available options, and a synopsis of the command syntax.

On-line Manual

More detailed information concerning various commands is normally available using the UNIX **man** command. Currently, there is no on-line help available for the Humdrum Toolkit so this *Reference Manual* will need to be a constant companion.

NAME

assemble — amalgamate two or more Humdrum files

SYNOPSIS

assemble *inputfile1 inputfile2* [*inputfile3* ...] [> *outputfile*]

DESCRIPTION

The **assemble** command allows two or more structurally similar Humdrum files to be aligned together — such as where a full score is assembled from files containing individual parts.

The **assemble** command is similar to the UNIX `paste` command. If file 'A' contains:

```
1
2
3
```

and file 'B' contains:

```
A
B
C
```

then the UNIX command:

```
paste A B
```

will produce the following output:

```
1	A
2	B
3	C
```

The **assemble** command is a more sophisticated version of `paste` — suitable for assembling several concurrent spines stored in different Humdrum files. The **assemble** command coordinates and synchronizes comments, interpretations, and data records from the input files. Duplicate global comments are avoided. Corresponding local comments are output where appropriate. Where one file contains local comments and a second file contains none, null local comments are inserted as appropriate. Similarly, null interpretations may also be added as necessary.

The **assemble** command expects that the input files will normally have the same number of data records. These data records are aligned side-by-side. If the input files do not contain the same number of data records, then the spines in the shorter file will be terminated with appropriate spine-path terminators.

OPTIONS

The **assemble** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

The operation of **assemble** can be illustrated by considering the following two input files. Both files contain the same number of data records (4).

file1:

```
!! A sample file.
**foo      **foo      **foo
!1         !2         !3
X          .          X
X          .          .
*          *v         *v
X          X
.          X
*-         *-
```

file2:

```
!! A sample file.
!! An additional global comment.
**bar        **bar
!4           !5
.            .
X            X
*v           *v
! joined
X
.
*-
```

If the **assemble** command is invoked as:

```
assemble file1 file2
```

then the corresponding output is:

```
!! A sample file.
!! An additional global comment.
**foo   **foo   **foo       **bar   **bar
!1      !2      !3          !4      !5
X       .       X           .       .
X       .       .           X       X
*       *v      *v          *       *
*       *       *v          *v
!       !       ! joined
X       X       X
.       X       .
*-      *-      *-
```

Notice that both input files begin with the identical global comment; only one copy of this comment appears in the output. The second file contains an additional global comment that is also output. The subsequent local comments have been amalgamated on a single output line. The spine-path changes in both input files have been properly rendered by padding null interpretations in the appropriate spines. The last local comment in `file2` has also been correctly re-positioned. Finally, each of the four data records have been aligned.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

census (4), **extract** (4), **humdrum** (4), **paste** (UNIX), **timebase** (4)

LIMITS

The number of specified input files may be limited on DOS systems.

NAME

census — determine general properties of a Humdrum input

SYNOPSIS

census [**-k**] [*inputfile* ...]

DESCRIPTION

The **census** command provides a summary of seven gross features of any Humdrum input. It provides counts of the total number of records (lines), the number of unique interpretations encountered, the number of comments, the number of data records, the number of data tokens, null tokens, and multiple-stops.

When the **-k** option is invoked, **census** provides a summary of a further ten features pertaining to `**kern` inputs. This summary includes the number of single and double barline records, the maximum number of concurrent notes, the total number of notes, the total number of rests, the number of untied notes, as well as the longest, shortest, highest, and lowest notes encountered.

OPTIONS

The **census** command provides the following options:

-h displays a help screen summarizing the command syntax
-k also output information regarding `**kern`-related data

Options are specified in the command line.

The **-k** option pertains to `**kern` inputs only. This option adds `**kern`-related information to the output.

SAMPLE OUTPUT

The following is a sample output where the **-k** option has been invoked. Without the **-k** option, the "KERN DATA" would be absent from the output.

```
HUMDRUM DATA
Number of data tokens:       33
Number of null tokens:       9
Number of multiple-stops:    1
Number of data records:      11
Number of interpretations:   2
Number of records:           14

KERN DATA
Number of notes:             8
Number of untied notes:      7
Longest note:                2
Shortest note:               16
Highest note:                cc
Lowest note:                 C
Number of rests:             4
Maximum concurrent notes:    2
Number of single barlines:   2
Number of double barlines:   1
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****kern** (2), **humdrum** (4), **proof** (4)

NAME

cents — translate pitch-related representations to cents

SYNOPSIS

cents [**-p** *n*] [**-tx**] [*inputfile ...*] [> *outputfile.cnt*]

DESCRIPTION

The **cents** command transforms various pitch-related inputs to corresponding numerical values in hundredths of semitones. It outputs one or more Humdrum `**cents` spines containing values corresponding to the cents distance from middle C for pitch-related input tokens. Pitches above middle C produce positive output values, whereas pitches below middle C produce negative output values. For example, the `**pitch` token "C3" is transformed to -1200 (cents).

The **cents** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**cents`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **cent** command should be given names with the distinguishing '.cnt' extension.

`**cents`	hundredths of a semitone with respect to middle C=0
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 equals C5)
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **cents**.

OPTIONS

The **cents** command provides the following options:

- **-h** displays a help screen summarizing the command syntax
- **-p** *n* output precision of *n* decimal places
- **-t** suppresses printing of all but the first note of a group of tied `**kern` notes
- **-x** suppresses printing of non-cents signifiers

Options are specified in the command line.

The **-p** option can be used to set the precision of the output values to *n* decimal places. The default precision is integer values only (*n*=0). Note that **cents** is able to process `**cents` as input; this feature allows the user to round-off existing `**cents` data to a specified precision.

The **-t** ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **cents** outputs non-pitch-related signifiers in addition to the cents value. For example, the `**pitch` token "A5zzz" will result in the output "2100zzz" — that is, after translating A5 to 2100 cents, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "8aa"; after translating 'aa' to 2100 cents, the non-pitch-related signifier '8' will also be output, hence the value 82100 — which will undoubtedly cause confusion. The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **cents.** The input contains six pitch-related spines — two of which (`**deg` and `**cocho`) cannot be processed by **cents.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'cents' example.
**kern  **pitch   **MIDI     **deg   **metpos  **cocho    **Tonh    **embell
*M2/4   *M2/4     *M2/4      *M2/4   *M2/4     *M2/4      *M2/4     *M2/4
*       *         *          *       *tb8      *          *         *
=1      =1        =1         =1      =1        =1         =1        =1
8ee-    G#4foo    /60/bar    1foo    1         r          Gis2      ct
.       .         /-60/      .       .         .          .         .
8ff     A3        /62/       2       3         9.89       H2        upt
.       .         /-62/      .       .         .          .         .
8dd-    Ab3       /70/       1       2         7.07       B2        ct
.       .         /-70/      .       .         .          .         .
8d-     C#4       /61/       6       3         7.135      Cis4      sus
.       .         /-61/      .       .         .          .         .
=2      =2        =2         =2      =2        =2         =2        =2
[4a-    r         .          5       1         r          r         .
.       .         .          7       3         5.5        Heses2    ct
4a-]    D4        /48/ /52/  1       2         8.11       C3        ct
.       .         /-48/      .       .         .          .         .
.       D4 F4     /-52/      2       3         7.33 6.4   C3 Es3    ct
=3      =3        =3         =3      =3        =3         =3        =3
r       G4        .          r       1         r          H2 D3     .
==      ==        ==         ==      ==        ==         ==        ==
*-      *-        *-         *-      *-        *-         *-        *-
```

Executing the command

```
cents -tx input > output.cnt
```

produces the following result:

```
!! 'cents' example.
**cents   **cents   **cents      **deg   **metpos   **cocho     **cents        **embell
*M2/4     *M2/4     *M2/4        *M2/4   *M2/4      *M2/4       *M2/4          *M2/4
*         *         *            *       *tb8       *           *              *
=1        =1        =1           =1      =1         =1          =1             =1
1500      800       0            1foo    1          r           -1600          ct
.         .         .            .       .          .           .              .
1700      -300      200          2       3          9.89        -1300          upt
.         .         .            .       .          .           .              .
1300      -400      1000         1       2          7.07        -1400          ct
.         .         .            .       .          .           .              .
100       100       100          6       3          7.135       100            sus
.         .         .            .       .          .           .              .
=2        =2        =2           =2      =2         =2          =2             =2
800       r         .            5       1          r           r              .
.         .         .            7       3          5.5         -1500          ct
.         200       -1200 -800   1       2          8.11        -1200          ct
.         .         .            .       .          .           .              .
.         200 500   .            2       3          7.33 6.4    -1200 -900     ct
=3        =3        =3           =3      =3         =3          =3             =3
r         700       .            r       1          r           -1300 -1000    .
==        ==        ==           ==      ==         ==          ==             ==
*-        *-        *-           *-      *-         *-          *-             *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the **kern spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the **pitch, **MIDI, and **cocho spines have been stripped away (due to the **-x** option).

FILES

The file x_option.awk is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), ****freq** (2), **freq** (4), ****fret** (2), ****kern** (2), **kern** (4), ****MIDI** (2), **midi** (4), ****pitch** (2), **pitch** (4), ****semits** (2), **semits** (4), ****solfg** (2), **solfg** (4), ****specC** (2) **specC** (4), ****Tonh** (2), **tonh** (4)

NAME

cleave — join tokens from two or more spines into a single output spine

SYNOPSIS

cleave [**-r**] [**-d** *delim*] **-i** '***in_interp1* [,***in_interp2* ...]' **-o** '***out_interp*' [*inputfile* ...]

DESCRIPTION

The **cleave** command permits concurrent data tokens in two or more specified spines to be amalgamated into a single data token and output in a single new spine. Consider, for example, two input spines containing pitch information and duration information respectively; **cleave** can be used to form a new spine that combines the pitch and duration signifiers into a single representation.

When the **cleave** command is invoked, the user identifies the input spines to be combined and specifies the name of the resulting output spine. Only a single output spine can be generated by **cleave.**

If necessary, the user may specify delimiter characters that are inserted between the component parts of the combined data token. (See EXAMPLES below.)

The **cleave** command amalgamates all spines containing the interpretation(s) specified by the user. For example, the command:

```
cleave -i '**recip,**kern' -o '**output' katsumi
```

will amalgamate all spines in the file `katsumi` containing `**recip` or `**kern` data — and will output the amalgamated data in a spine labelled `**output`.

Note that the output spine generated by **cleave** preserves the same record-type structure as the input, and so may readily be pasted with the input file using the Humdrum **assemble** command.

The **cleave** command is able to adapt to spine-path changes throughout the input. When a processed input spine is *split,* the new spine participates in the amalgamated output spine. When a processed spine is *exchanged,* **cleave** continues to track its location. When a new spine is *added,* to the input it is included in the amalgamated output only if its interpretation matches one of the interpretations being processed. When a processed spine is *terminated,* **cleave** continues to process other spines in the input; if no target spine remains, then null tokens are generated until the end of the input or until other target interpretations appear. *Join*-spine indicators have no affect on the output.

Notice that the **cleave** command is useful for transforming a spine that periodically splits and joins into a single spine containing multiple-stops.

OPTIONS

The **cleave** command provides the following options:

-d *delimiter*	interpose the string *delimiter* between amalgamated tokens
-i *'in_interp'*	list of input interpretations to be processed
-o *'out_interp'*	specify output interpretation
-r	suppress outputting of duplicate (repeated) signifiers

Options are specified in the command line.

The **-d** option is used to specify a string that is inserted between each component part forming the assembled output token. The default delimiter is the null string.

A given signifier (character) may be present in two or more concurrent input tokens — such as the letter 'A' shared by the tokens 'AB' and 'AX'. Depending on the task, the user may want only a single instance of each signifier to be echoed in the output (e.g. 'ABX' rather than 'ABAX'). The **-r** option causes characters that are common to two or more input tokens to be output only once. The output position of any repeated character corresponds to the first instance of the character in the processed input.

EXAMPLES

The following examples illustrate the operation of the **cleave** command.

Consider the following input consisting of four spines representing octave-class, diatonic pitch letter name, accidental, and cents-deviation:

```
**oct   **diaton   **accid   **Cdev
4       G          .         .
4       A          .         .
4       B          b         -10
5       C          .         .
5       D          .         .
5       E          b         .
5       F          #         +12
5       G          .         .
*-      *-         *-        *-
```

The information available in these four spines might be amalgamated into a single spine by executing the following command:

```
cleave -i '**diaton,**accid,**oct,**Cdev' -o '**pitch' input
```

The following output would be produced:

```
**pitch
G4
A4
Bb4-10
C5
D5
Eb5
F#5+12
G5
*-
```

The output interpretation has been specified as `**pitch`. Notice that the order of the signifiers in each output data token reflects the order of the input interpretations given in the command line — i.e., `**diaton` values first, followed by `**accid`, followed by `**oct`, followed by `**Cdev`. In the case of `**accid` and `**Cdev` data, notice that null tokens (periods) do not affect the output token. In the default invocation, note that no intervening characters are placed between the joined subtokens.

In the following example, **cleave** is used to create double-stops from two spines having identical interpretations. Notice the presence of barlines.

```
**kern	**kern
*M2/4	*M2/4
*foo	*bar
=1	=1
4c	4e
8d	8f
8e	8g
=2	=2
8f	8a
8g	8b
8a	8cc
8b	8dd
=3	=3
4cc	4ee
*-	*-
```

Executing the command

```
cleave -i '**kern' -d ' ' -o '**kern' input > output
```

will produce the following output:

```
**kern
*M2/4
*
=1 =1
4c 4e
8d 8f
8e 8g
=2 =2
8f 8a
8g 8b
8a 8cc
8b 8dd
=3 =3
4cc 4ee
*-
```

Notice that if identical tandem interpretations appear in the target spines, then they are echoed in the output. Otherwise a null interpretation is output.

The redundant measure numbers in the above output might be eliminated using the following **humsed** command:

```
humsed '/^=/s/ .*//' input > output
```

Alternatively, the input might have been preprocessed so that the barlines in one of the two input spines were replaced by null tokens.

The **-r** option can be used to eliminate duplicate or repeated signifiers. Consider, for example, the following input:

```
**kern    **kern
.         4c
(8        8d
)8        8e
(         8f
)         8g
8'        8a
8'        8b
.         4cc
*-        *-
```

The first `**kern` spine includes articulation information not present in the second spine. The pitch, duration, and articulation information can be amalgamated without duplication of the duration information using the **-r** option:

```
cleave -r -i '**kern' -o '**kern' input > output
```

The resulting output is:

```
**kern
4c
(8d
)8e
(8f
)8g
8'a
8'b
4cc
*-
```

Users should be careful when using the **-r** option while at the same time assigning a delimiter that appears in the input stream. For example, if the slash (/) is defined as an output delimiter, and the **-r** option is invoked, then following input:

```
ab    a/b
```

will produce the following output:

```
ab//
```

Note that the first slash in the above output delimits the material originally contained in the left and right spines. The second slash is a bona fide signifier in the right spine. If the delimiter in the above example was a space rather than a slash, then the result would produce trailing spaces — and so the output would no longer conform to the Humdrum syntax.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

assemble (4), **extract** (4), **humsed** (4), **rend** (4)

WARNINGS

Syntactically correct Humdrum output is not guaranteed if the **-r** option is invoked while using the space as a delimiter.

The use of regular expression metacharacters as delimiters (such as ^) can cause problems for **cleave.**

NAME

context — congeal data records to form a contextual frame

SYNOPSIS

context [**-b** *regexp*] [**-d** *string*] [**-e** *regexp*] [**-i** *regexp*] [**-n** *n*] [**-o** *regexp*] [**-p** *n*] [*inputfile* ...]

DESCRIPTION

The **context** command amalgamates one or more successive input data records into single records according to user-defined criteria. Only single-spine Humdrum inputs are permitted. The **context** command provides a useful means for amalgamating on a single line those data tokens that are somehow deemed to be contextually related. For example, **context** might be used to link together all pitches in a measure, or pair the first and last notes of each phrase. The command is useful in such tasks as partitioning possible pitch-class sets or grouping arpeggio tones into chords.

In its simplest mode of operation **context** will join a specified number of successive data records together to form a single output record. By way of example, consider a file (named `input`) consisting of a single spine whose data records contain the numbers 1 through 6 on separate lines:

```
**numbers
1
2
3
4
5
6
*-
```

The command

```
context -n 3 input
```

will produce the following output:

```
**numbers
1 2 3
2 3 4
3 4 5
4 5 6
.
.
*-
```

Notice that the output file has been padded with null tokens so that the number of output records matches the number of input records. By invoking the **-p** option, the padded null tokens can be placed either at the beginning or the end of the file, or split between beginning and end. For example, the command

```
context -n 3 -p 3 input
```

will place the trailing null tokens in the above example at the beginning of the output. The **-p** option defaults to the value 0.

If null tokens are present in the input, they remain in place, yet do not affect the congealed data records. For example, if a null token was present between the numbers 1 and 2 in the above input, the command

```
context -n 4 -p 1 input
```

would produce an output beginning with a single padded null token:

```
**numbers
.
1 2 3 4
.
2 3 4 5
3 4 5 6
.
.
*-
```

Rather than specifying a fixed number of congealed data records, input records can alternatively be amalgamated according to the signifiers present in the input data itself. The **-e** option allows the user to specify an "end" signifier. When this signifier is encountered in the input, the input record is appended to the current congealed record — which is then output — and a new congealed output record begins. End signifiers are defined as string patterns using the *regular expression* syntax (see **regexp (5)**). For example, given an input of six successive numbers, the command

```
context -e [0246] input
```

would produce the following output:

```
**numbers
1 2
.
3 4
.
5 6
.
*-
```

OPTIONS

The **context** command supports the following options:

-b *regexp*	begin a new output record starting with token matching *regexp*
-d *string*	use *string* as output delimiter for input records rather than the space character
-e *regexp*	begin a new output record starting after token matching *regexp*
-h	displays a help screen summarizing the command syntax
-i *regexp*	ignore any records matching *regexp* when counting
-n *n*	amalgamate *n* input data records for each output record
-o *regexp*	omit any records matching *regexp* from amalgamated output; do not count
-p *n*	pad *n* (normally trailing) null tokens at the beginning of the output spine

Options are specified in the command line. Note that the **-b** and **-e** options are mutually exclusive with the **-n**, **-p**, and **-i** options.

In the default operation, **context** separates amalgamated tokens by inserting a space character. (Thus the input tokens are treated as subtokens in a Humdrum multiple-stop.) The **-d** option allows the user to define an alternative string as the subtoken delimiter.

The **-n** option allows the user to specify the maximum number of data records assembled into a single output record.

The **-b** option allows the user to specify a "begin" marker. When this marker (*regexp*) is matched in the input, any current congealed record is output, and a new congealed record begins. Begin signifiers are defined as string patterns using the *regular expression* syntax.

With the **-e** option, if **context** encounters a data record matching *regexp* then it appends the current input record to the current assembled output record and begins assembling a new record with the following input record.

The **-i** option is used only with **-n**; it causes any data records matching *regexp* to be ignored in the counting of amalgamated tokens. Such "uncounted" records are nevertheless output.

The **-o** option causes data records matching *regexp* to be omitted from the output.

The **-p** option may be used in conjunction with **-n**. Normally, the output from **context -n** is padded with trailing null tokens — one fewer in number than the value specified with **-n**. The **-p** *n* option causes *n* null tokens to be padded at the *beginning* of the output spine, rather than trailing at the end.

Note that tandem interpretations and comments are processed like null tokens; they are merely echoed in the output in their appropriate position. Note also that **context** automatically breaks a congealed output record whenever it encounters a spine-path terminator or exclusive interpretation in the input.

EXAMPLES

The following excerpt from Edgar Varèse's *Density 21.5 (1936)* illustrates the use of **context.** Consider the initial input:

```
!! Edgar Varèse, Density 21.5 (1936)
!! excerpt: mm.41-45
**kern
*MM72
=41
(16f#
16e#
[8gn
2.g_
=42
4g])
4r
4r
8r
(16f#
16e#
=43
6gn)
(6e#
6f#
8g)
(8f#
12e#
12g
12dn
=44
2a-)
(4an
8een
[8bb-
=45
4bb-]
2.ccc#)
=45
8r
*-
```

A simple transformation would be to amalgamate successive data records in overlapping groups of 3. The following command:

```
context -n 3 density
```

would produce the following output:

```
!! Edgar Varèse, Density 21.5 (1936)
!! excerpt: mm.41-45
**kern
*MM72
=41 (16f# 16e#
(16f# 16e# [8gn
16e# [8gn 2.g_
[8gn 2.g_ =42
2.g_ =42 4g])
=42 4g]) 4r
4g]) 4r 4r
4r 4r 8r
4r 8r (16f#
8r (16f# 16e#
(16f# 16e# =43
16e# =43 6gn)
=43 6gn) (6e#
6gn) (6e# 6f#
(6e# 6f# 8g)
6f# 8g) (8f#
8g) (8f# 12e#
(8f# 12e# 12g
12e# 12g 12dn
12g 12dn =44
12dn =44 2a-)
=44 2a-) (4an
2a-) (4an 8een
(4an 8een [8bb-
8een [8bb- =45
[8bb- =45 4bb-]
=45 4bb-] 2.ccc#)
4bb-] 2.ccc#) =45
2.ccc#) =45 8r
.
.
*-
```

Notice once again that the input and output have the same number of records. Preserving the structure in this way allows the user to coordinate the contextual output with the original input using the **assemble** command.

A more useful transformation might amalgamate successive data records in overlapping groups of 3 *notes;* that is rests and barlines should be ignored. The following command causes input records containing either an equals-sign or the letter 'r' to be ignored when counting the number of amalgamated data records:

```
context -n 3 -i [=r] density
```

The input and corresponding output are given in the left and right spines below:

```
!! Edgar Varèse, Density 21.5 (1936)
!! excerpt: mm.41-45
**kern    **kern
*MM72     *MM72
=41       =41 (16f# 16e# [8gn
(16f#     (16f# 16e# [8gn
16e#      16e# [8gn 2.g_
[8gn      [8gn 2.g_ =42 4g])
2.g_      2.g_ =42 4g]) 4r 4r 8r (16f#
=42       =42 4g]) 4r 4r 8r (16f# 16e#
4g])      4g]) 4r 4r 8r (16f# 16e#
4r        4r 4r 8r (16f# 16e# =43 6gn)
4r        4r 8r (16f# 16e# =43 6gn)
8r        8r (16f# 16e# =43 6gn)
(16f#     (16f# 16e# =43 6gn)
16e#      16e# =43 6gn) (6e#
=43       =43 6gn) (6e# 6f#
6gn)      6gn) (6e# 6f#
(6e#      (6e# 6f# 8g)
6f#       6f# 8g) (8f#
8g)       8g) (8f# 12e#
(8f#      (8f# 12e# 12g
12e#      12e# 12g 12dn
12g       12g 12dn =44 2a-)
12dn      12dn =44 2a-) (4an
=44       =44 2a-) (4an 8een
2a-)      2a-) (4an 8een
(4an      (4an 8een [8bb-
8een      8een [8bb- =45 4bb-]
[8bb-     [8bb- =45 4bb-] 2.ccc#)
=45       .
4bb-]     .
2.ccc#)   .
=45       .
8r        .
*-        *-
```

Notice that as the end of the file is approached, **context** will continue amalgamating data records until it is no longer able to satisfy the amalgamating criteria. If unable to complete an output record, **context** will output a null token.

If the above command had used the **-o** rather than the **-i** option, all of the rests and barlines would have been omitted from the output. Otherwise, the output would be the same as given above.

A more musically useful partitioning of Varèse's work might be based on slur markings. The following command uses open- and closed-slur markings to demarcate the contextual outputs; (note the need to escape the parentheses since they are regular expression metacharacters).

```
context -b '\(' -e '\)' -o '[=r]' density
```

Notice that the ensuing output (shown below) pads the output with null tokens in order to maintain the same number of data records as the original input. (An output such as the following might be used as input to a command sequence such as **pc -x | pcset**).

```
!! Edgar Varèse, Density 21.5 (1936)
!! excerpt: mm.41-45
**kern
*MM72
.
(16f# 16e# [8gn 2.g_ 4g])
.
.
.
.
.
.
.
.
(16f# 16e# 6gn)
.
.
.
(6e# 6f# 8g)
.
.
(8f# 12e# 12g 12dn 2a-)
.
.
.
.
.
(4an 8een [8bb- 4bb-] 2.ccc#)
.
.
.
.
.
.
.
*-
```

If there were any notes not embraced within a slur, the above command would have caused them to be output on their own line.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

context (4), **grep** (UNIX), **nf** (4), **patt** (4), **pattern** (4), **pcset** (4)

WARNINGS

The **-b** and **-e** options are mutually exclusive with the **-n**, **-p**, and **-i** options.

NAME

correl — measure the numerical similarity between two spines

SYNOPSIS

correl [**-f** *templatefile*] [**-m**] [**-p** *n*] [**-s** *regexp*] [*inputfile* ...] [> *outputfile.cor*]

DESCRIPTION

The **correl** command measures the degree of parametric (numerical) similarity between corresponding values in two Humdrum spines. More precisely, **correl** calculates Pearson's coefficient of correlation for paired tokens containing numerical data.

Two modes of operation are provided. In the *single input mode*, a single file containing two equal-length spines is processed. In this mode, the output from **correl** consists of a single number indicating the linear correlation between the two spines of numerical data.

In the *dual input mode*, two single-spine numerical inputs — called the *template file* and the *input file* — are specified by the user. Normally, the template file is considerably shorter than the input file. In this mode, the output consists of a spine of numerical information (`**correl`) that reflects the momentary similarity between the template and the input for each successive moment in the input. In short, the input file is 'scanned' using the template values, and the correlational similarity determined at each point.

In both *single input* and *dual input* modes, output numerical values range between +1 and -1. Correlation values reflect the degree to which two sets of numerical values rise and fall in synchrony. The maximum output value is +1 — indicating that the two sets of numbers are perfectly related according to a linear relationship. A minimum output value of −1 indicates that the two sets of numbers are perfectly out-of-phase — one set of numbers rises while the other set falls by a proportional magnitude, and vice versa. A correlation value of zero indicates that there is no linear relationship between the two sets of numbers.

In *single input mode*, inputs to **correl** must consist of precisely two spines; otherwise an error message is generated and the command is terminated. The two spines may contain different interpretations and represent different types of information. In the case of the *dual input mode*, the input file and template file must have precisely one spine each; the spines may differ in length, but the *template* file must not be longer than the *input* file.

Only numerical signifiers are considered by **correl;** non-numeric input data are ignored. Where a data token contains a mix of numeric and non-numeric signifiers, only the first complete numerical subtoken contributes to the calculation. The following examples illustrate how **correl** interprets mixed data tokens:

data token	numerical interpretation
`4gg#`	4
`4.gg#`	4
`-33aa`	-33
`-aa33`	33
`x7.2yz`	7.2
`a7..2bc`	7
`[+5]12`	5
`$17@2`	17
`a1b2 c.3.d`	1 0.3

Humdrum multiple-stops require special attention in **correl** (see below).

In the *dual input mode* the output from the **correl** command consists of a set of records matching the structure of the *input* document. Output values indicate the correlation between the *template* data values and the input data values beginning at that record.

When the *dual input mode* is invoked, it is recommended that output files produced by the **correl** command should be given names with the distinguishing '.cor' extension.

OPTIONS

The **correl** command provides the following options:

-f *templatefile*	specify source pattern as *templatefile* and invoke dual input mode
-h	displays a help screen summarizing the command syntax
-m	disable matched-pairs criterion
-p *n*	output precision to *n* decimal places
-s *regexp*	skip; completely ignore data records matching *regexp*

Options are specified in the command line.

The **-f** option is used to specify an independent *template file* and so invoke the *dual input mode.*

The **-p** option can be used to set the precision of the output values to *n* decimal places. The default precision is 3 decimal places.

The **-s** option allows the user to avoid (or skip) the processing of certain types of data records. This option must be accompanied by a user-defined regular-expression. Input data records matching this expression are not processed.

Correlation values can be calculated only where all numerical data are arranged as matched pairs — that is, the input conforms to the "matched pairs criterion." For example, the following two spines illustrate numerical data matching. The number of numerical data values in both spines are matched throughout the inputs:

```
**spine1   **spine2
10.0       4
7 3        2 .91
.          .
13.8       4
5 8 5      1 1 2
a b c      x
.          p q
*-         *-
```

By contrast, the following file shows several transgressions of the matched pairs criterion. For example, the first data record gives a numerical value in spine #1 that is not matched by a numerical value in spine #2. Similarly, the multiple-stop values in the second data record are unmatched in spine #2:

```
**spine1   **spine2
9.7        a
7 31       2
.          114
426        .
r 11 7     35 xy08z 28
a b c      6 .07
.          p q
*-         *-
```

In normal operation, a single failure to conform to the matched pairs criterion will cause **correl** to issue an error message and terminate operation. If the **-m** option is invoked, unmatched data is simply ignored. For example, with the **-m** option, the above input is treated as equivalent to the following input:

```
**spine1   **spine2
.          .
7          2
.          .
.          .
11 7       35 08
.          .
.          .
*-         *-
```

EXAMPLES

The following examples illustrate the operation of **correl.** The first example shows an excerpt containing considerable parallel motion between two polyphonic voices. Measuring the pitch-contour similarity can be done using the single input mode.

```
!! J.S. Bach, Invention No. 8; BWV 779
**semits        **semits
*M3/4           *M3/4
9               17
12              21
10              19
12              21
9               17
12              21
10              19
12              21
=6              =6
5               14
9               17
7               16
9               17
*-              *-
```

In order to avoid processing the measure numbers, the skip (**-s**) option is used; executing the command:

```
correl -s = bwv779
```

will produce the following output:

```
0.979
```

The second example illustrates the dual input mode. The target input consists of a single spine (labelled `**input`) containing mixed alphabetic and numerical values. (This input file is shown below as the left-most spine.) The template file consists of the numerical sequence: 1, 2, 3 — mixed with the letters a, b, c. (This file is shown as the middle spine below.) Note that the non-numeric characters in both the input and template files have no influence on the operation of **correl.** The third (output) spine is produced by the following command:

```
correl -f template input > output.cor
```

```
(input      (template    (correl
 file)        file)      output)
**input     **template   **correl
0           1a           1.000
1           2b           1.000
2           3c           1.000
3           *-           -0.655
4                        -0.655
x1x                      0.866
y2.                      0.866
2z                       0.000
(3)                      -1.000
[2]                      .
01                       .
*-                       *-
```

The similarity values generated by **correl** are given in the `**correl` spine. Each successive value in the output spine is matched with a data token in the target input file (`**foo`). For example, the initial three output values (1.000) indicate that exact positive correlations occur between the template and the input. That is (0, 1, 2) (1, 2, 3) and (2, 3, 4) all show simple equi-distant increases corresponding to the source template. The final numerical value in `**correl` shows a negative correlation (-1.000) indicating that the numerical sequence (3, 2, 1) is the exact opposite contour to the source template (1, 2, 3). By contrast, the immediately preceding output value (0.000) indicates that the sequence (2, 3, 2) shows no systematic linear relationship with the source template (1, 2, 3).

The following example provides a more complicated illustration of **correl.** Once again the left-most spine is the target input, the middle spine is the source template, and the right-most spine shows the corresponding output.

```
(input      (template    (correl
 file)        file)      output)
**input     **template   **correl
=1          1            .
1           2 3          1.000
2 3         .            -0.370
100         4            -0.742
8r          5 6          .
4           *-           0.042
5 6                      .
=2                       .
0                        .
4r                       .
-2x -3                   .
-x8                      .
==                       .
*-                       *-
```

The above output spine was created by executing the command:

```
correl -m -s '[=r]' -f template input > output.cor
```

Due to the **-s** option, all records in the input file containing an equals-sign or lower-case 'r' are eliminated from the calculations. The presence of the null-token in the third data record of the template file is noteworthy. Although no correlations are calculated with the null-token, it acts as a place-holder, and causes the corresponding record in the input file to be ignored. For example, the first correlation value is calculated on the basis of the following coordination of numerical data:

```
1      1
2 3    2 3
100    .
4      4
5 6    5 6
```

Since the value '100' is not matched with a numerical value in the template, it is ignored in the correlation measure. (Note that without the **-m** option, no output would be generated.)

At the next instant, the correlation value is calculated on the basis of the following coordination of numerical data:

```
2 3    1
100    2 3
4      .
5 6    4
0      5 6
```

The double-stops do not form matched pairs, hence much of the data is discarded. For example, in the first data record, 2 is matched with 1 and 3 is discarded. In the second record, 100 is matched with 2 and 3 is discarded, etc.

The third correlation value is calculated on the basis of the following coordination of numerical data:

```
100      1
4        2 3
5 6      .
0        4
-2 -3    5 6
```

In this case, the correlation value is based on the following numerical pairing: 100 ⇔ 1, 4 ⇔ 2, 0 ⇔ 4, –2 ⇔ 5, –3 ⇔ 6. All other numerical values are ignored.

The final correlation value in this example is calculated on the basis of the following coordination of numerical data:

```
4        1
5 6      2 3
0        .
-2 -3    4
8        5 6
```

The corresponding correlation value is based on the following numerical pairing: 4 ⇔ 1, 5 ⇔ 2, 6 ⇔ 3, –2 ⇔ 4, 8 ⇔ 5.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

patt (4), **pattern** (4), **simil** (4)

WARNINGS

Correlation coefficients indicate only the magnitude of the association between two sets of data. High correlation values can occur purely by chance. The noteworthiness (statistical significance) of a correlation value depends on the number of input values given in the template spine. Novice users should consult a standard statistics textbook for further advice on how to interpret the results.

For formal statistical measures, the **-m** option should never be invoked.

If only one pair of matched values is present, the linear correlation is mathematically undefined. In this case a question mark signifier is output.

LIMITS

The **correl** command is currently unable to handle input files greater than about 4,000 records.

NAME

deg — translate pitch-related representations to relative scale degree (`**deg`)

SYNOPSIS

deg [**-tx**] [*inputfile* ...] [> *outputfile.deg*]

DESCRIPTION

The **deg** command transforms various pitch-related inputs to the corresponding scale degree. The command outputs one or more Humdrum `**deg` spines — where scale degrees are indicated by the numbers 1 (tonic) to 7 (leading tone). Scale degree information can be determined only with reference to some prevailing key. For example, the pitch C4 is the tonic (1) in the key of C major, but the submediant (6) in the key of E minor. The **deg** command expects a tandem interpretation indicating the key of the input passage; **deg** will adapt to specified changes of key within an input. If no key information is provided prior to the first pitch-related data, **deg** issues an error message and terminates.

The **deg** command differs from the (related) **degree** command in that it outputs *relative* (rather than *absolute)* pitch-height information. Upward pitch motions are indicated by the caret (^), whereas downward pitch motions are indicated by the lower-case letter 'v'. Hence, the token '1' followed by '^5' means that the ensuing dominant pitch is above rather than below the preceding tonic pitch. No absolute pitch-height information is represented. As in the case of **degree,** plus and minus signs indicate whether a pitch has been chromatically raised or lowered. For example, the pitch A-flat is designated '6-' in the key of C major, but '6' in the key of C minor. The harmonic minor scale is assumed for all minor keys. Thus, B-flat is considered a "lowered" seventh degree in C minor, whereas B natural is considered the "normal" (rather than "raised") seventh degree. For some applications, this interpretation of the minor-scale seventh degree may cause difficulties.

The **deg** command is able to translate any of the pitch-related representations listed below. For descriptions of the various input representations (including `**deg`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **deg** command should be given names with the distinguishing '.deg' extension.

`**kern`	core pitch/duration representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**solfg`	French solfège system (fixed 'doh')
`**Tonh`	German pitch system

Input representations processed by **deg.**

OPTIONS

The **deg** command provides the following options:

-h displays a help screen summarizing the command syntax
-t suppresses printing of all but the first note of a group of tied `**kern` notes
-x suppresses printing of non-`**deg` signifiers

Options are specified in the command line.

The **-t** option ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **deg** outputs non-pitch-related signifiers in addition to the degree value. For example, in the key of D major, the `**pitch` token "G5zzz" will result in the output "4zzz" — that is, after translating G5 to 4, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "4f#" in the key of D minor; after translating '`f#`' to '`3+`' (i.e. raised third degree), the preceding non-pitch-related signifier '`4`' will also be output, hence the value `43+` — which may cause confusion.

The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **deg.** The input contains four pitch-related spines — one of which (`**MIDI`) cannot be processed by **deg.** In addition, there is one non-pitch-related spines (`**embell`).

```
!! 'deg' example.
**kern   **Tonh   **MIDI      **solfg  **pitch  **embell
*M2/4    *M2/4    *M2/4       *M2/4    *M2/4    *M2/4
*C:      *d:      *G#:        *a:      *F:      *F:
=1       =1       =1          =1       =1       =1
8ee-     Gis2     /60/        do3      F4foo    ct
.        .        /-60/       .        .        .
8f       H2       /62/        fa3      r        upt
.        .        /-62/       .        .        .
8dd-     B2       /70/        mi3      E4       ct
.        .        /-70/       .        .        .
8d--     Cis4     /61/        r        F4       sus
.        .        /-61/       .        .        .
=2       =2       =2          =2       =2       =2
[4a-     r        .           mi~b3    F4 A4    .
.        Heses2   .           re3      G4 Bb4   ct
4a-]     C3       /48/ /52/   do3      E4 C5    ct
.        .        /-48/       .        .        .
.        H2 E3    /-52/       la3      G4       ct
=3       =3       =3          =3       =3       =3
r        A2 F3    .           r        F4       .
==       ==       ==          ==       ==       ==
*-       *-       *-          *-       *-       *-
```

Executing the command:

```
deg -tx input > output.deg
```

produces the following result:

```
!! 'deg' example.
**deg   **deg    **MIDI      **deg   **deg   **embell
*M2/4   *M2/4    *M2/4       *M2/4   *M2/4   *M2/4
*C:     *d:      *G#:        *a:     *F:     *F:
=1      =1       =1          =1      =1      =1
3-      4+       /60/        3       1       ct
.       .        /-60/       .       .       .
v4      ^6+      /62/        ^6      r       upt
.       .        /-62/       .       .       .
^2-     v6       /70/        v5      v7      ct
.       .        /-70/       .       .       .
v2-     ^7       /61/        r       ^1      sus
.       .        /-61/       .       .       .
=2      =2       =2          =2      =2      =2
^6-     r        .           v5-     1 ^3    .
.       v6-      .           v4      v2 ^4   ct
.       ^7-      /48/ /52/   v3      v7 ^5   ct
.       .        /-48/       .       .       .
.       v6+ ^2   /-52/       ^1      v2      ct
=3      =3       =3          =3      =3      =3
r       v5 ^3    .           r       v1      .
===     ===      ===         ===     ===     ===
*-      *-       *-          *-      *-      *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**pitch` spine have been stripped away (due to the **-x** option). Note that the plus and minus signs merely indicate that a scale degree has been raised or lowered, but not by how much. Hence both the D-flat and D double-flat in measure 1 of the first (`**kern`) spine are rendered as '`2-`'.

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****deg** (2), ****degree** (2), **degree** (4), ****kern** (2), **kern** (4), ****pitch** (2), **pitch** (4), ****solfg** (2), **solfg** (4), ****Tonh** (2), **tonh** (4)

NAME

degree — translate pitch-related representations to absolute scale degree (`**degree`)

SYNOPSIS

degree [**-tx**] [*inputfile* ...] [> *outputfile.dgr*]

DESCRIPTION

The **degree** command transforms various pitch-related inputs to the corresponding scale degree. The command outputs one or more Humdrum `**degree` spines — where scale degrees are indicated by the numbers 1 (tonic) to 7 (leading tone). Scale degree information can be determined only with reference to some prevailing key. For example, the pitch C4 is the tonic (1) in the key of C major, but the submediant (6) in the key of E minor. The **degree** command expects a tandem interpretation indicating the key of the input passage; **degree** will adapt to specified changes of key within an input. If no key information is provided prior to the first pitch-related data, **degree** issues an error message and terminates.

The **degree** command differs from the (related) **deg** command in that it outputs *absolute* (rather than *relative)* pitch-height information. Along with the scale degree, the octave number is represented — the two values being separated by a slash (/). Hence the token '1/4' means the first scale degree (tonic) in octave '4'. As in the case of **deg,** plus and minus signs indicate whether a pitch has been chromatically raised or lowered. For example, the pitch A-flat is designated '6-' in the key of C major, but '6' in the key of C minor. The harmonic minor scale is assumed for all minor keys. Thus, B-flat is considered a "lowered" seventh degree in C minor, whereas B natural is considered the "normal" (rather than "raised") seventh degree. For some applications, this interpretation of the minor-scale seventh degree may cause difficulties.

The **degree** command is able to translate any of the pitch-related representations listed below. For descriptions of the various input representations (including `**degree`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **dgr** command should be given names with the distinguishing '.dgr' extension.

`**kern`	core pitch/duration representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**solfg`	French solfège system (fixed 'doh')
`**Tonh`	German pitch system

Input representations processed by **degree**.

OPTIONS

The **degree** command provides the following options:

- **-h** displays a help screen summarizing the command syntax
- **-t** suppresses printing of all but the first note of a group of tied `**kern` notes
- **-x** suppresses printing of non-`**degree` signifiers

Options are specified in the command line.

The **-t** option ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **degree** outputs non-pitch-related signifiers in addition to the degree value. For example, in the key of D major, the `**pitch` token "G5zzz" will result in the output "4/5zzz" — that is, after translating G5 to 4/5, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "4f#" in the key of D minor; after translating '`f#`' to '`3+/4`' (i.e. raised third degree in octave 4), the preceding non-pitch-related signifier '`4`' will also be output, hence the value `43+/4` — which may cause confusion.

The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **degree.** The input contains four pitch-related spines — one of which (`**MIDI`) cannot be processed by **degree.** In addition, there is one non-pitch-related spines (`**embell`).

```
!! 'degree' example.
**kern   **Tonh    **MIDI       **solfg   **pitch   **embell
*M2/4    *M2/4     *M2/4        *M2/4     *M2/4     *M2/4
*C:      *d:       *G#:         *a:       *F:       *F:
=1       =1        =1           =1        =1        =1
8ee-     Gis2      /60/         do3       F4foo     ct
.        .         /-60/        .         .         .
8f       H2        /62/         fa3       r         upt
.        .         /-62/        .         .         .
8dd-     B2        /70/         mi3       E4        ct
.        .         /-70/        .         .         .
8d--     Cis4      /61/         r         F4        sus
.        .         /-61/        .         .         .
=2       =2        =2           =2        =2        =2
[4a-     r         .            mi~b3     F4 A4     .
.        Heses2    .            re3       G4 Bb4    ct
4a-]     C3        /48/ /52/    do3       E4 C5     ct
.        .         /-48/        .         .         .
.        H2 E3     /-52/        la3       G4        ct
=3       =3        =3           =3        =3        =3
r        A2 F3     .            r         F4        .
==       ==        ==           ==        ==        ==
*-       *-        *-           *-        *-        *-
```

Executing the command:

```
degree -tx input > output.dgr
```

produces the following result:

```
!! 'degree' example.
**degree    **degree    **MIDI      **degree    **degree    **embell
*M2/4       *M2/4       *M2/4       *M2/4       *M2/4       *M2/4
*C:         *d:         *G#:        *a:         *F:         *F:
=1          =1          =1          =1          =1          =1
3-/5        4+/2        /60/        3/3         1/4         ct
.           .           /-60/       .           .           .
4/4         6+/2        /62/        6/3         r           upt
.           .           /-62/       .           .           .
2-/5        6/2         /70/        5/3         7/4         ct
.           .           /-70/       .           .           .
2-/4        7/4         /61/        r           1/4         sus
.           .           /-61/       .           .           .
=2          =2          =2          =2          =2          =2
6-/4        r           .           5-/3        1/4 3/4     .
.           6-/2        .           4/3         2/4 4/4     ct
.           7-/3        /48/ /52/   3/3         7/4 5/5     ct
.           .           /-48/       .           .           .
.           6+/2 2/3    /-52/       1/3         2/4         ct
=3          =3          =3          =3          =3          =3
r           5/2 3/3     .           r           1/4         .
==          ==          ==          ==          ==          ==
*-          *-          *-          *-          *-          *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**pitch` spine have been stripped away (due to the **-x** option). Note that the plus and minus signs merely indicate that a scale degree has been raised or lowered, but not by how much. Hence both the D-flat and D double-flat in measure 1 of the first (`**kern`) spine are rendered as degree '`2-`'.

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****deg** (2), **deg** (4), ****degree** (2), ****kern** (2), **kern** (4), ****pitch** (2), **pitch** (4), ****solfg** (2), **solfg** (4), ****Tonh** (2), **tonh** (4)

NAME

encode — interactive Humdrum encoding from MIDI input

SYNOPSIS

encode [**-r** *controlfile.rc*] [*editfile*]

DESCRIPTION

The **encode** command provides an interactive editor for capturing Humdrum data from a MIDI input device. The operation of **encode** can be tailored by the user to generate a wide range of Humdrum representations — including representations designed by the user.

The **encode** command is limited to encoding information one spine at a time. A typical use of **encode** might be to encode individual musical "parts" or "voices" in a representation such as `**kern`; the user might then use the Humdrum **timebase** and **assemble** commands to construct a full score from the encoded parts. The user must have access to an appropriate MIDI controller connected via a Roland MPU-401 (or compatible) MIDI controller card. Currently, **encode** is available only for IBM PC or compatible hardware running under the DOS operating system.

The **encode** command implements a full-screen interactive text editor — similar to the **vi** text editor. When invoked, the screen is divided into three display regions: a *status window,* a *command window,* and a larger *text window.* The *status window* displays various pieces of information, including the name of the file being edited, the current size of the file (in bytes), the run-control (`.rc`) file used to configure the editor, the number of beats per measure, the output signifier assigned to each beat, the current metronome rate, and mode. The *command window* is used to construct and execute general-purpose commands. The *text window* is used to display and edit the encoded Humdrum text.

The editor has four modes of operation: *edit mode, input mode, MIDI input mode,* and *command mode.*

EDIT MODE

When **encode** is first invoked, the editor is placed in *edit mode.* If an edit-file has been specified in the command-line, then the beginning of any existing text is displayed in the text window. Edit mode allows the user to move to particular locations in the text and execute certain types of text manipulation instructions. Movement through the text can be achieved by either scrolling the display, or by relative cursor motions within the text window. The cursor position is indicated by a pulsing underscore character.

Four scrolling commands are available. **Control-F** is used to scroll forward in the document by one page. **Control-B** is used to scroll backward by one page. **Control-D** is used to scroll down by half a page. **Control-U** is used to scroll up by half a page. The **page-down** and **page-up** keys can be used instead of CTRL-F and CTRL-B respectively. If any of these commands is preceded by a typed *number*, then the action is repeated *number* times. Hence 5<*page-down*> will cause the display to advance five pages.

Movement to specific line numbers can be achieved using the upper-case '**G**' command preceded by the desired line number. If no line number is specified, the **G** command causes the cursor to be moved to the end of the file.

Vertical motions of the cursor relative to the displayed text can be controlled by a variety of means. The up (↑) and down (↓) cursor-control keys move the cursor up one line or down one line respectively. Alternatively, the lower-case letters '**k**' and '**j**' may be used for up and down movements — as in the **vi** text editor. If the cursor is at the bottom or top of the screen, cursor movement down or up a line (respectively) will cause the screen to scroll one line as needed. The carriage return or **ENTER** key will cause the cursor to move to the beginning of the next line. Once again, if any of these commands is preceded by a typed *number,* then the action is repeated *number* times.

The upper-case letter '**H**' positions the cursor at the highest line displayed on the screen, whereas the upper-case letter '**L**' positions the cursor at the lowest line displayed on the current screen. Alternatively, the '**home**' and '**end**' keys can be used to move the cursor to the top and bottom respectively. The upper-case letter '**M**' positions the cursor in the middle of the current screen. In all of these commands, the cursor is positioned at the horizontal beginning of the appropriate line.

Horizontal motions of the cursor within a line can be carried out using the left (←) and right (→) arrow keys. Alternatively, the lower-case '**h**' and '**l**' keys can be used as aliases for the left and right cursor motions — as in the **vi** text editor. The **space** bar can also be used to move the cursor to the right. The dollar sign (**$**) causes the cursor to move to the end of the current line. The number zero (**0**) or the caret (^) causes the cursor to move to the beginning of the current line.

Horizontal cursor movements may also be defined by context. Where multiple data tokens are encoded on a single line, the '**w**' command causes the cursor to move forward word-by-word, whereas the '**b**' command causes backward movement word-by-word. A *word* is defined as any character string separated by spaces (or by new lines). In both the '**w**' and '**b**' commands the cursor is placed at the beginning of the appropriate word.

Three forms of simple text deletions are possible in *edit mode.* The character corresponding to the current cursor position can be deleted by typing the lower-case letter '**x**'. The character immediately before the current cursor position can be deleted by typing the upper-case letter '**X**'. The lower-case letter '**d**' will cause the current line to be deleted; line deletions may also be achieved by typing the '**Del**' key. Preceding any of these commands by a typed *number* causes the action to be repeated *number* times. The most recent deletion can be restored or "undone" by typing the lower-case letter '**u**'.

INPUT MODE

The *input mode* allows the manual inputting of typed characters into the text file. This mode is invoked from the *edit mode* by typing either '**a**', '**i**', '**o**', or '**O**'. When any of these commands is invoked, the cursor increases in size and the status window reports the new mode. Typing the letter '**i**' will cause all subsequently typed material to be *inserted* immediately prior to the current cursor position. Typing the letter '**a**' will cause all subsequently typed material to be *appended* following the current cursor position. This can also be achieved by typing the '**Ins**' or '**INSERT**' key. Typing the letter '**o**' will cause a new line to be *opened* following the current line; subsequently typed material will begin on this new line. Typing the upper-case letter '**O**' will cause a new line to be opened immediately *prior* to the current line.

Once the input mode is invoked, typed ASCII characters will be added at the current cursor position until the ESCape or TAB key is pressed. Pressing ESCape or TAB will return **encode** to the *edit mode.* Tabs are explicitly forbidden as input in order to reduce the possibility of creating text that does not conform to the Humdrum syntax.

MIDI INPUT MODE

The most characteristic feature of **encode** is the *MIDI input mode.* The MIDI input mode can be invoked from *edit mode* by typing either of the upper-case letters '**A**' or '**I**'. The letter '**I**' will cause all subsequent MIDI-driven input to be *inserted* immediately prior to the current line. The letter '**A**' will cause all subsequent MIDI-driven input to be *appended* immediately following the current line.

In the *MIDI input mode,* data is entered into the current document according to user-defined mappings between *MIDI events* (such as caused by playing on a MIDI keyboard) and ASCII character strings. Predefined mappings are specified in a *run-time control* (`.rc`) file. When the **encode** command is invoked, it must have access to a `.rc` file containing configuration information. The user can identify a specific `.rc` file on the command line via the **-r** option (see OPTIONS). Alternatively, **encode** will seek a default `.rc` file (named `encode.rc`) residing in the the current directory, or if absent, in the `$HUMDRUM/etc` directory. The run-time control file is essential to the operation of **encode,** and inability to locate a `.rc` file will cause **encode** to abort (see discussion in OPTIONS below).

The run-control file contains a series of definitions mapping MIDI events to output strings. Three classes of MIDI events can be mapped: *key number* (`KEY`), *delta-time* (`DEL`), and *key velocity* (`VEL`). Normally, a `.rc` file contains a number of definitions for each class of event, although some events may not appear at all in some `.rc` files.

By way of example, the run-control instruction:

```
KEY 60 middle-C
```

assigns the key-on event for MIDI key #60 to the string `middle-C`. When in *MIDI input mode,* each key-on event for key #60 will cause the string `middle-C` to be prepared for insertion into the text window. Typically, a number of `KEY` definitions will appear in a given run-control file — often one definition for each MIDI key (0 to 127).

A second class of MIDI events is key-down velocity (`VEL`). Key velocities can range between 0 (low key velocity) and 127 (high key velocity). Each `VEL` mapping specifies a range of values for which a given string will be output. By way of example, the following assignment maps key velocity values between 90 and 127 to a string consisting merely of the apostrophe (the `**kern` signifier for a staccato note):

```
VEL 90 127 '
```

Given this mapping, key-down velocities within the specified range will cause the apostrophe character to be prepared for insertion into the text window.

A third class of MIDI events is delta-time (`DEL`). When determing the "duration" of a performed note, the durations of individual key-presses are confound by the articulation. In general, performing musicians are less concerned by the *duration* of individual key-presses, than by the keypress-to-keypress time spans; the elapsed time between one key-onset and the next key-onset provides a better estimate of the nominal musical duration of a note than the actual held duration. The variable `DEL` contains the *difference between successive key-onset times* — expressed in MIDI clock ticks. Values of `DEL` may range from 0 upward. For a tempo of 60 beats per minute, inter-onset durations of one second correspond to `DEL` values of about 100.

The following sample `.rc` file illustrates a simple run-control file. Notice that a series of `DEL` ranges have been defined and mapped to `**kern`- or `**recip`-type durations. For example, inter-onset times lying between 48 and 80 clock ticks generate the output string '`8`'; values between 113 and 160 generate the string '`4`' and so on. Notice that this file restricts the number of possible output "durations" to just five.

```
# Sample .rc file
KEY 60 c
KEY 62 d
KEY 64 e
KEY 65 f
KEY 67 g
KEY 69 a
KEY 71 b
KEY 72 cc

DEL 48 80 8
DEL 81 112 8.
DEL* 113 160 4
DEL 161 224 4.
DEL 225 320 2

VEL 90 127 '

ORDER DEL KEY VEL
```

Any records in the run-control file beginning with a # character are treated as comments. Empty lines are ignored.

The effect of the above run-control file can be illustrated by example. Imagine that **encode** received two key-on events (key #60 followed by key #62), where the first key exhibited a velocity value of 94 and the inter-onset time (`DEL`) was 100. The first key (#60) would be mapped to the string '`c`'; the delta-time would be mapped to the string '`8.`'; and the key-velocity (`VEL`) would be mapped to the apostrophe. At the moment of the key-onset for key #62, these three strings would be amalgamated according to the `ORDER` instruction (`DEL` first, `KEY` second, and `VEL` third) — producing the output string: `8.c'`

Notice that **encode** outputs assembled strings only when the *next* key-on event occurs. This means that the text display is always one "note" behind the performer. Note that if musical durations are based on key inter-onset times, it is impossible to output a note prior to the onset of the next note. The last note in a buffer can be flushed by typing the ESCape key. (The timing of the last note is based on the `DEL` between key-onset and the moment of pressing ESCape.)

In addition to mapping velocities, inter-onset times, and key numbers, run-control files can define a number of other attributes. The MIDI channel number attended to by **encode** can be set by the `RECEIVE` instruction. Any one of 16 channels (1-16) can be selected. A default channel 1 is assumed if no `RECEIVE` instruction is present in the run-control file.

The **encode** command has a built-in metronome for assisting real-time encoding. The metronome sends commands to the MIDI instrument generating metronome tones. Two types of tones are generated — tones marking each beat, and tones marking the beginning of each measure. The metronome rate (in beats per minute) is set by the TEMPO command. The beat is specified in two ways. The default beat is indicated by the presence of an asterisk following *one* of the DEL instructions. In the above example, the signifier '4' is assigned to the default beat. Apart from the default beat, the beat may also be explicitly assigned using the BEAT instruction. This instruction is followed by a single argument identifying the output *signifier* intended to coincide with each metronome beat. For example, `BEAT 4.` would set the beat in the above example to the dotted quarter, rather than the quarter. Note that the specified signifier in the BEAT command must correspond to one of the existing signifiers defined using a DEL instruction.

The following table lists all of the types of instructions permitted in a run-control file.

# *text*	unexecutable comment
BEAT *string*	set beat to **DEL** whose output signifier is *string* beat
BUFFER *n text*	define (potentially multi-line) text buffer # *in* (0-9)
DEL *min max string*	key inter-onset times between *min* and *max* clock ticks cause *string* to be output
KEY *n string*	MIDI key-on #*n* causes *string* to be output
METER *n*	define number of beats per measure as *n*
METRE *n*	same as **METER**
MM *on*|*off*	switch metronome *on* or *off*; default is *on*
ORDER *codeword1 codeword2 ...*	define order of string outputs, where codewords are selected from: BUFFER, DEL, KEY, SRING, VEL
RECEIVE *n*	define the MIDI channel from which data is accepted
STRING *n text*	define string constant # *in* (0-9)
TEMPO *n*	set metronome to *n* beats per minute
VEL *min max string*	key-down velocities between *min* and *max* cause *string* to be output

Definition types for **encode**

The `BUFFER`, `DEL`, `KEY`, `STRING`, and `VEL` instructions can be repeated multiple times within the `.rc` file. All other instructions (`BEAT`, `METER`, `MM`, `TEMPO`, `ORDER` and `RECEIVE`) should appear only once in the `.rc` file. The `BEAT` and `TEMPO` instructions cannot appear in the `.rc` file until after the default beat (`DEL*`) has been defined.

COMMAND MODE

The **encode** *command mode* allows a number of general-purpose commands to be executed — such as editing a specified file, changing a default mapping, or auditioning the encoded material. The *command mode* can be invoked from the **encode** *edit mode* by typing the colon character (:).

Each command is formulated in the *command window* and launched by pressing the ENTER key. After execution, **encode** is returned to *edit mode.*

Most commands consist of a single character; some commands require one or more parameters.

The '**w**' command causes the current text to be written to disk. If there is currently no active filename, then an error is displayed indicating that **encode** is unable to write a file without knowledge of the filename. The command '**w** *filename*' will cause the current contents to be written to the file *filename*. If **encode** was invoked with a specified filename, then that filename is active throughout the session.

If the user attempts to write to an existing file (that was not specified when **encode** was invoked), then an error message is issued. Overwriting an existing file can be achieved by appending an exclamation mark following the write instruction — as in '**w!** *filename*.'

The '**q**' command causes **encode** to terminate. If the current file has been modified without writing to disk, then a warning will be displayed and the quit instruction ignored. Appending an exclamation mark ('**q!**') will cause **encode** to terminate without saving any recent modifications.

Note that the *quit* and *write* commands can be combined as a single instruction — '**wq**'.

The '**r** *filename*' command causes **encode** to read the file *filename* into the text, beginning at the line following the current cursor position.

The '**v**' command causes **encode** to spawn a **vi** text editing session — importing the current **encode** text. The **vi** text editor provides text manipulation capabilities, including searching, substitution, and macro-instruction facilities not available in **encode.** (Refer to the UNIX **vi** reference document for further information.) When the **vi** session is closed, the edited text file is returned to the **encode** session.

The '**m**' command invokes a Humdrum pipeline that is suitable for auditioning text data conforming to either the `**kern` or `**MIDI` representation formats. Specifically, '**m**' causes the current text to be passed to the Humdrum pipeline: `midi -c | perform`. Any `**kern` data will be translated to `**MIDI` data and sent to the `perform` command. The user can then interactively proof-listen or audition the encoded data. Refer to the **perform** **(4)** for information regarding the types of interactive commands available during proof-listening. The **perform** command is terminated when the end-of-file is reached, or if the user presses either the ESCape key or the letter 'q'. In either case, control is returned to **encode.**

The '**b**' command is used to read buffer text defined in the `.rc` file. Up to ten numbered buffers (0-9) can be defined. The command:

```
b 1
```

will cause any text denoted `BUFFER 1` to be output following the current cursor position. Use of the upper-case '**B**' rather than '**b**' causes the buffer contents to be inserted prior to the current line rather than following the current line.

Buffer zero (0) has a special status. When the **encode** command is invoked, if the current text is empty (i.e. empty file or no filename specified), then the contents of `BUFFER 0` are automatically inserted into the text. This provides a convenient way to import header information for a newly encoded file.

The '**rc** *filename*' command causes **encode** to use a different run-control file *filename*. This allows the **encode** environment to be entirely reconfigured without interrupting the encoding session. This command can prove useful, for example, when the music being encoded changes key.

The **set** command can be used to define (or redefine) any parameters permitted in a `.rc` file. For example, the tempo may be changed, the metronome turned-off, the metronome beat redefined, a string variable assigned, or a specific key re-mapped, e.g.

```
set TEMPO 92
set MM off
set BEAT 4.
set STRING 3 !! Variation No. ...
set KEY 60 C4
```

For the `BUFFER` command, **set** defines an *additional* buffer record, rather than replacing any existing buffer definitions.

Note that run-control keywords, such as `TEMPO`, may be spelled using either upper-case or lower-case.

Note that due to the small size of the command window, especially long command lines may wrap around within the window. This wrap-around has no affect on the command operation.

SUMMARY

The various built-in commands in **encode** are summarized in the following table.

EDIT MODE

CTRL-F	scroll forward one page
CTRL-B	scroll backward one page
CTRL-D	scroll down by half a page
CTRL-U	scroll up by half a page
<PAGE-DOWN>	scroll forward one page
<PAGE-UP>	scroll backward one page
↓	move cursor down one line
↑	move cursor up one line
j	move cursor down one line
k	move cursor up one line
<ENTER>	move cursor to the beginning of the next line
H	move cursor to the top of the display
M	move cursor to the middle of the display

L	move cursor to the bottom of the display
<HOME>	move cursor to the top of the display
<END>	move cursor to the bottom of the display
→	move cursor one character to the right
←	move cursor one character to the left
l	move cursor one character to the right
h	move cursor one character to the left
<SPACE>	move cursor one character to the right
x	delete character at current cursor position
X	delete character immediately preceding currrent cursor position
d	delete current line
u	undo most recent deletion or insertion command
<INSERT>	insert text prior to current cursor position (invoke INPUT MODE)
i	insert text prior to current cursor position (invoke INPUT MODE)
a	insert text after to current cursor position (invoke INPUT MODE)
o	insert text beginning with the next line (invoke INPUT MODE)
O	insert text beginning with the previous line (invoke INPUT MODE)
A	invoke MIDI INPUT MODE; insert data beginning with the next line
I	invoke MIDI INPUT MODE; insert data beginning with the previous line
<*number*>	repeat ensuing command <*number*> times

INPUT MODE

ESC	return to EDIT MODE
<TAB>	return to EDIT MODE

MIDI INPUT MODE

ESC	complete last MIDI event and return to EDIT MODE

COMMAND MODE

b *n*	append run-control buffer number *n* following current cursor position
B *n*	insert run-control buffer number *n* before current cursor position
m	invoke interactive proof-listening for `**kern` or `**MIDI` text
set *rc-command*	set or reset a `.rc` mapping
q	quit **encode**
q!	quit **encode** without saving modifications since last write
r *filename*	read input file *filename* at current cursor position
rc *controlfile*	use run-control file *controlfile* rather than current `.rc` file
s *n*	append run-control string number *n* following current cursor position
S *n*	insert run-control string number *n* before current cursor position
v	spawn **vi** text editing session using current text
w [*filename*]	write file *filename* to disk; defaul filename is current file
w! *filename*	overwrite existing file *filename*
wq	write current file and quit

OPTIONS

The **encode** command provides the following options:

-h displays a help screen summarizing the command syntax
-r *file.rc* invoke using the run-control file *file.rc*

Options are specified in the command line.

The **-r** option permits the user to identify a specific run-control file to configure the **encode** editor. If this option is omitted, **encode** will seek a default run-control file named `encode.rc` in the current directory, or in the directory `$HUMDRUM/etc` if not present in the current directory. If the option is specified, **encode** will search the current directory for the specified run-control file. If this search fails to locate the file, **encode** will search `$HUMDRUM/etc`. If this fails, **encode** will treat the input filename as an absolute file path. If this fails, encode will issue an error message indicating that it failed to locate the specified run-control file.

FILES

A number of predefined `.rc` files are maintained in the `$HUMDRUM/etc` directory. Exploration is encouraged. The default file is `$HUMDRUM/etc/encode.rc`.

PORTABILITY

DOS 2.0 and up. An appropriate MIDI controller (such as a keyboard synthesizer) connected via a Roland MPU-401 (or compatible) MIDI controller card. The `vi` text editor must be available in order to invoke the 'v' edit command.

WARNINGS

Unlike the UNIX **vi** text editor, only a single '**d**' is required in edit mode to delete a line (rather than two d's). Experienced **vi** users should be careful when deleting lines.

Note that the `BEAT` and `TEMPO` instructions cannot appear in the `.rc` file until after the default beat (`DEL*`) has been defined.

SEE ALSO

assemble (4), **cleave** (4), **encode.rc** (5), **humdrum** (4), ****kern** (2), ****MIDI** (2), **midi** (4), **num** (4), **perform** (4), **proof** (4), **record** (4), **timebase** (4), **vi** (UNIX)

REFERENCES

Use the the Music Quest Inc. MIDI library functions is gratefully acknowledged.

NOTE

Especially long input lines may exceed the size of the text window. Although the characters to the right of the text screen may not be visible, they remain encoded in the file.

NAME

extract — get specified spines from a Humdrum input

SYNOPSIS

extract -f *field1,field2,...,$-1,$* [*inputfile* ...]
extract -i *interp1,interp2,...,interp*n [*inputfile* ...]
extract -p *spine#*n [*inputfile* ...]
extract -t *field_trace_file.ftf* [*inputfile* ...]

DESCRIPTION

The **extract** command allows the user to select one or more spines from a Humdrum input. The command is typically used to extract parts (such as a tuba part) from some multi-part score; however **extract** can be used to isolate dynamic markings, musical lyrics, or any other stream of information that has been encoded as a separate Humdrum spine.

The **extract** command has four distinct modes of operation: (1) field mode **(-f)**, (2) interpretation mode **(-i)**, (3) spine-path mode **(-p)**, and (4) field-trace mode **(-t)**. The simplest mode is field mode; the most general-purpose mode is field-trace mode.

In *field mode*, the **extract** command operates in a manner similar to the UNIX `cut` command. The user may specify a given field (data column) or set of fields to be selected from the input stream. For example, the command:

```
extract -f 1,3,8
```

will extract the first, third, and eighth spines from the input stream. In field mode, field specifications may also be made with respect to the right-most field. For example, the expression `"$"` refers to the right-most field in the input; the arithmetic expression `"$-1"` refers to the right-most field minus one, etc. By way of example, the command:

```
extract -f '2,4-6,$' lassus
```

will extract the second, fourth, fifth, sixth, and last (right-most) spines in the file "lassus." The `extract -f` command differs from the UNIX `cut` command in that Humdrum global comments are properly preserved in the output.

In *interpretation mode*, the **extract** command outputs all spines containing the interpretation(s) specified by the user. By way of example, the command:

```
extract -i '**semits,**MIDI' hildegard
```

will extract all spines in the file `hildegard` containing `**semits` or `**MIDI` data.

In *spine-path mode,* the **extract** command follows a given spine starting at the beginning of the file, and traces the course of that spine throughout the input stream. If spine-path changes are encountered in the input (such as spine-exchanges, spine-merges, or spine-splits) the output adapts accordingly. By way of example, if the "third" spine is selected, the output consists of the third spine and follows the path of that spine through the input until it is terminated or the end-of-file is encountered. What begins as the third column, may end up as some other column (or columns) in the input.

In the *field-trace mode,* the **extract** command accepts a list of spine-column positions over the course of the file. In the *field-trace* mode, the user provides a file containing a list specifying the precise selection of spines through the file at various line numbers. The first column in this list specifies the line (record) number at which the field selection is defined. The second column lists the spine or spines to be extracted beginning at the specified line. For example, the following *field-trace file* specifies that the first spine from line 1 should be output; beginning at line 2, spines 3, 4, and 5 should be output; beginning at line 18, spines 1 and 4 should be output, and from line 78 to the end of the input, spine 9 should be output.

```
1     1
2     3-5
18    1,4
78    9
```

The *field-trace* mode allows the user to select virtually any combination of data tokens from the input stream. Note that using the *field-trace* mode may produce output that no longer conforms to the Humdrum syntax. (See EXAMPLES below.)

OPTIONS

The **extract** command provides the following options:

-f *fieldlist*	select field mode
-h	displays a help screen summarizing the command syntax
-i *interplist*	select interpretation mode
-p *spine#*	select spine-path mode
-t *fieldtrace file*	select field-trace mode

Options are specified in the command line.

A *fieldlist* consists of any set of integers separated by commas, or a range specification in which the lower and upper values are separated by a dash: e.g.`1,4-8,13`. With the exception of range specifications, the order of the integers in the list is unimportant, hence `13,1,4-8` is equivalent to `1,4-8,13`. The right-field anchor ($) can be used only with the **-f** option.

An *interplist* consists of any set of tandem or exclusive interpretations separated by commas, e.g.`'**kern,*C:'` Option arguments may require the use of quotation marks in order to prevent inadvertent expansion of shell metacharacter (*, ?, etc.).

EXAMPLES

The following examples illustrate how **extract** may be used.

```
extract -f '1,3,$' holst
```

outputs the first, third, and last columns from the file `holst`.

```
extract -p 4 mossolov
```

outputs the spine *beginning* (but not necessarily continuing) in the fourth column of the file `mossolov`.

```
extract -t sibelius.fld sibelius
```

outputs the spines specified in the file `sibelius.fld` for the file `sibelius`. (See SAMPLE OUTPUTS below.)

```
extract -i '*F:,*f:' hendrix
```

outputs all spines that contain the tandem interpretations *F: or *f: (i.e. in the keys of F major or F minor).

SAMPLE OUTPUTS

The following examples illustrate the various **extract** options. Consider the following input file, dubbed `input1`:

```
!! 'extract' example #1
**ABC    **xyz    **123    **ABC    **foo
A        x        1        a        bar
B        y        2        b        .
C        z        3        c        #
*-       *-       *-       *-       *-
```

Executing either of the following commands:

```
extract -f '1,3,$-1' input1 > output
```

or

```
extract -i '**ABC,**123' input1 > output
```

would produce the following result:

```
!! 'extract' example #1
**ABC     **123     **ABC
A         1         a
B         2         b
C         3         c
*-        *-        *-
```

Consider next the following sample input — dubbed `input2`:

```
!! 'extract' example #2
**ABC     **xyz     **123     **ABC     **foo
A         x         1         a         %
*         *         *^        *         *
B         y         2a        2b        b         &
C         z         3a        3b        c         #
*         *x        *x        *         *         *
A         4a        xyz       4b        d         %
*-        *         *         *         *-        *
5a        xyz       5b        &
*         *x        *x        *
6a        6b        xyz       #
*v        *v        *         *
7         xyz       %
8         xyz       &
*-        *-        *-
```

Executing the command:

```
extract -p 3 input2 > output
```

would produce the following result:

```
!! 'extract' example #2
**123
1
*^
2a            2b
3a            3b
*x            *
4a            4b
*             *
5a            5b
*             *x
6a            6b
*v            *v
7
8
*-
```

Notice that this output no longer conforms to the Humdrum syntax. (Output lines 7 and 11 contain only a single exchange-path interpretation.)

For the 'example #2' input file, the field-structure is as follows:

```
1    1-1   # Line 1 must appear in the file.
4    1-5   # *     *     *^     *     * ....
5    1-6   # Line after path indicator record
7    1-6   # *     *x    *x     *       ....
8    1-6   # Line after path indicator record
9    1-6   # *-    *     *      *     * ....
10   1-4   # Line after path indicator record
11   1-4   # *     *x    *x     *       ....
12   1-4   # Line after path indicator record
13   1-4   # *v    *v    *      *       ....
14   1-3   # Line after path indicator record
16   1-3   # *-    *-    *-             ....
```

(The above file may be generated via the `fields -s` command.) On the basis of this information a user might create the following field-trace file, dubbed `trace`:

```
1    1
4    3
5    3,5
7    2,3
8    3,2
9    1,3
10   4
14   3
15   2
16   1
```

Executing the following command:

```
extract -t trace input2 > output
```

would produce the following result:

```
!! 'extract' example #2
**ABC
A
*^
2a          b
3a          c
*x          *x
4a          xyz
*-          *
&
*
#
*
%
xyz
*-
```

Notice that in this case, data tokens have been selected from a variety of input spines.

DIAGNOSTICS

In *field-trace* mode, if the specified field-structure does not correspond to the actual input file, then an ERROR message is issued.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

WARNINGS

Note that only the *field* mode (**-f**) guarantees that the output will conform to the Humdrum syntax.

In *interpretation* mode, tandem interpretations may be specified in the command invocation, but mutually exclusive tandem interpretations (such as changes of key) will not force the program to stop outputting a specified spine.

SEE ALSO

assemble (4), **cut** (UNIX), **fields** (4), **yank** (4)

NAME

fields — list spine/field structure of a Humdrum file

SYNOPSIS

fields [**-gils**] [**-r** *regexp*] [*inputfile* ...] [> *outputfile.ftf*]

DESCRIPTION

The **fields** command works in conjunction with the **-t** option of the **extract** command to permit highly selective extraction of data from a Humdrum file. This command is used only rarely; it is invoked when the more usual options of **extract** fail to provide sufficient flexibility in isolating certain Humdrum data.

The **fields** command outputs a so-called "field-trace file" for a given Humdrum input. This file can be edited by the user and then used with the **extract** command to select data from the original input file. For repetitive tasks, the field-trace file is more convenient than manual editing of the input file.

When invoking the **fields** command, the user specifies the types of records to be used as reference points in the editing task. The command then produces a listing of record numbers as well as the corresponding number of fields for each record of the specified type. For example, the user may wish to use spine-path changes as reference points for editing an input.

Each output record from the **fields** command consists of three pieces of information, such as illustrated below:

```
13      1-10      # {(4.g#  4.b  ...
```

The first field consists of a single number identifying the corresponding line number of the input file. The second field consists of two numbers separated by a dash. The second number indicates the number of currently active spines. The number prior to the dash is always 1 (see below). The third field is a comment beginning with the octothorpe (#) and continuing with the first ten characters of the corresponding line from the input file. The purpose of this comment information is to help orient the user when editing a field-trace file.

Field-trace information can be requested by record-type: **-g** for global comments, **-l** for local comments, **-i** for exclusive and tandem interpretations, and **-s** for spine-path indicators. More than one record-type can be requested. For example, when the **-gl** options are invoked, the **fields** command will produce an output line each time a local or global comment is encountered in the input. The line number and number of fields will be given in the output.

For global comments, the number of fields output for the current line is equal to the number of fields for the most recent non-global comment record. For spine-path records, the current record as well as the next record are output so that the user knows the changes in the number of spines.

A **-r** option permits the user to specify a *regular expression;* field-trace data is output for each record matching the specified regular expression.

Note that when an unedited field-trace file is used in conjunction with **extract -t,** the output is identical to the input. By modifying the field-range (second column in the output), the user can select which specific fields will be output.

It is recommended that output files produced using the **fields** command should be given names with the distinguishing '.ftf' extension.

OPTIONS

The **fields** command provides the following options:

-g	identify lines with global comments
-h	displays a help screen summarizing the command syntax
-i	identify any interpretation record
-l	identify lines with local commens
-r *regexp*	identify all lines matching *regexp*
-s	identify lines with spine-path indicators

Options are specified in the command line.

SAMPLE OUTPUTS

Consider by way of illustration the following input file:

```
!! 'fields' example
**numbers    **alpha     **symbols
*            *betics     *
14           abc         %@
!some        !local      !comments
3            .           #&
*-           *           *
jkl          $+
!! a global comment
*^           *
uvw          xyz         &%
*            *+          *
*            *           **numbers    *
mno          pqr         87           {^}
*-           *-          *-           *-
```

With the **-g** option, the fields command will produce the following output:

```
1   1-1   # Line 1 must appear in the file.
9   1-2   # !! a globa ....
```

With the **-l** option, the fields command will produce the following output:

```
1   1-1   # Line 1 must appear in the file.
5   1-3   # !some    !loc ....
```

With the **-s** option, the fields command will produce the following output:

```
1    1-1   # Line 1 must appear in the file.
7    1-3   # *-    *    * ....
8    1-2   # Line after path indicator record
10   1-2   # *^    * ....
11   1-3   # Line after path indicator record
12   1-3   # *     *+   * ....
13   1-4   # Line after path indicator record
15   1-4   # *-    *-   *-   * ....
```

With the **-i** option, the fields command will produce the following output:

```
1    1-1   # Line 1 must appear in the file.
2    1-3   # **numbers   ....
3    1-3   # *     *betics    ....
7    1-3   # *-    *    * ....
8    1-2   # Line after path indicator record
10   1-2   # *^    * ....
11   1-3   # Line after path indicator record
12   1-3   # *     *+   * ....
13   1-4   # *     *    **numb ....
15   1-4   # *-    *-   *-   * ....
```

Using the **-r** option we can specify a regular expression on which record information cues. Executing the following command:

```
fields -r '[0-9]' input
```

produces the following output:

```
1    1-1   # Line 1 must appear in the file.
4    1-3   # 14   abc   %@ ....
6    1-3   # 3    .     #& ....
14   1-4   # mno  pqr   87 ....
```

Notice that only those records containing numerical data tokens have been listed.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

extract (4), **regexp** (4), **regexp** (6)

NAME

fill — replace null tokens with previous data token

SYNOPSIS

fill [**-c** *chars*] [**-p**] [**-s** *regexp*] [*inputfile* ...]

DESCRIPTION

The **fill** command accepts any Humdrum input and replaces each null token with the previous non-null data token in the same spine.

Humdrum null-tokens are place-holders that do not themselves encode data. Null-tokens consist of a single period character (".") — separated from other tokens by tabs, or appearing on a line by itself. The **fill** command replaces occurrences of null-tokens with the most recent non-null data occurring in the same spine. When the **-p** option is invoked, the replacement data tokens are enclosed in parentheses ( ). If the initial data tokens in a spine are null-tokens, then null-tokens (.) are output.

In repeating previous data tokens, if the **-s** option is invoked, **fill** skips over any data records matching *regexp*. For example, if *regexp* is the equals-sign (the "common system" barline), then barline data tokens will not be repeated in subsequent data records containing null tokens. Thus, if a data token 'X' is followed by a token that matches the regular expression /=/, then subsequent null-tokens will be replaced by the token 'X' rather than by the equals sign.

The **fill** command correctly handles spine path changes such as exchange-path indicators (*x), join-path indicators (*v), split-path indicators (*^), terminate-path indicators (*-), and begin-spine (*+). In the case where two spines join together, **fill** outputs a double-stop where necessary.

OPTIONS

The **fill** command provides the following options:

-h	displays a help screen summarizing the command syntax
-c *chars*	repeats only characters listed in *chars*
-p	place repeated data tokens in parentheses
-s *regexp*	skip data records matching *regexp*

Options are specified in the command line.

EXAMPLES

The following inputs and outputs illustrate the operation of the **fill** command. Consider the following input:

```
!! Example 1
**kern	**kern
16e-	8r
16d	.
16e-	8gg
16f	.
16g	8cc
16f	.
16g	8gg
16e-	.
16a	[2aa
16g	.
16a	.
16b-	.
16cc	.
16b-	.
16cc	.
16a	.
=78	=78
.	.
*-	*-
```

Invoking the command:

```
fill input > output
```

produces the following output:

```
!! Example 1
**kern	**kern
16e-	8r
16d	8r
16e-	8gg
16f	8gg
16g	8cc
16f	8cc
16g	8gg
16e-	8gg
16a	[2aa
```

```
16g     [2aa
16a     [2aa
16b-    [2aa
16cc    [2aa
16b-    [2aa
16cc    [2aa
16a     [2aa
=78     =78
=78     =78
*-      *-
```

Notice that all of the null tokens have been replaced by the preceding data token in the same spine. Notice also that the barline for measure 78 has been repeated. For many applications repeating of barlines will be inappropriate.

The following, more complex example, illustrates the use of the **-p** and **-s** options. The input is shown on the left and the corresponding output is shown on the right:

INPUT			OUTPUT		
!! Example 2			!! Example 2		
**foo	**foo	**bar	**foo	**foo	**bar
a	xyz	.	a	xyz	.
.	23	(%&)	(a)	23	(%&)
=2	=2	=2	=2	=2	=2
.	.	.	(a)	(23)	((%&))
!! A comment.			!! A comment.		
.	.	49	(a)	(23)	49
*x	*	*x	*x	*	*x
.	.	.	(49)	(23)	(a)
*	*v	*v	*	*v	*v
.	.		(49)	(23 a)	
abc	XYZ		abc	XYZ	
*	*^		*	*^	
.	.	.	(abc)	(XYZ)	(XYZ)
.	1a	2b	(abc)	1a	2b
=3	=3	=3	=3	=3	=3
*-	*	*	*-	*	*
.	.		(1a)	(2b)	
====	====		====	====	
*+	*		*+	*	
**foo	**foo	**bar	**foo	**foo	**bar
.	.	.	(1a)	.	(2b)
*-	*-	*-	*-	*-	*-

The output was produced by invoking the following command:

```
fill -p -s ^= input > output
```

In order to avoid repeating the barlines, the skip option has been invoked with the regular expression "^=" — meaning any equals sign at the beginning of a line. (See **regexp** in Section 6 of this manual for details concerning regular expression syntax.) In addition, the **-p** option has been invoked so that all repeated tokens are placed in parentheses. Notice that **fill** adapts to changing spine-paths. Note especially the join-spine (*v) interpretations leading to the double-stop: `(23 a)`.

A final example illustrates the use of the **-c** option. Once again, the input is shown on the left and the corresponding output is shown on the right. The output was produced by invoking the following command:

```
fill -c '[a-gA-G#-]' input > output
```

INPUT		OUTPUT	
**kern	**kern	**kern	**kern
(4g	8b	(4g	8b
.	8cc	g	8cc
8f#	4dd	8f#	4dd
4.g)	.	4g)	dd
.	8cc	g	8cc
.	8b	g	8b
4d	4a	4d	4a
.	.	d	a
*-	*-	*-	*-

The effect of this command has been to propagate the `**kern` pitch signifiers, without propagating any non-pitch information.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

patt (4), **pattern** (4), **regexp** (4), **regexp** (6), **simil** (4)

NAME

freq — translate pitch-related representations to frequency

SYNOPSIS

freq [**-p** *n*] [**-tx**] [*inputfile* ...] [> *outputfile.frq*]

DESCRIPTION

The **freq** command transforms various pitch-related inputs to corresponding frequency representations. It outputs one or more Humdrum `**freq` spines containing numerical values (in hertz) corresponding to the fundamental frequency for pitch-related input tokens. For example, the `**pitch` token "C4" is transformed to 261.63 (hertz).

The **freq** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**freq`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **freq** command should be given names with the distinguishing '.frq' extension.

`**cbr`	critical band rate (in equivalent rectangular bandwidth units)
`**cents`	hundredths of a semitone with respect to middle C=0 (e.g. 1200 equals C5)
`**cocho`	cochlear coordinates (in millimeters)
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. -12 equals C3)
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **freq**.

OPTIONS

The **freq** command provides the following options:

- **-h** displays a help screen summarizing the command syntax
- **-p** *n* output precision of *n* decimal places
- **-t** suppresses printing of all but the first note of a group of tied `**kern` notes
- **-x** suppresses printing of non-frequency signifiers

Options are specified in the command line.

The **-p** option can be used to set the precision of the output values to *n* decimal places. The default precision is two decimal places. Note that **freq** is able to process `**freq` as **input**; this feature allows the user to round-off existing `**freq` data to a specified precision.

The **-t** option ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **freq** outputs non-pitch-related signifiers in addition to the frequency value. For example, the `**pitch` token "A6zzz" will result in the output "1760.00zzz" — that is, after translating A6 to 1760.00 hertz, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "8aaa"; after translating 'aaa' to 1760.00 hertz, the non-pitch-related signifier '8' will also be output, hence the value 81760.00 — which will undoubtedly cause confusion. The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **freq.** The input contains six pitch-related spines — one of which (`**deg`) cannot be processed by **freq.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'freq' example.
**kern	**pitch	**MIDI	**deg	**metpos	**cocho	**Tonh	**embell
*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4
*	*	*	*	*tb8	*	*	*
=1	=1	=1	=1	=1	=1	=1	=1
8ee-	G#4foo	/60/bar	1foo	1	r	Gis2	ct
.	.	/-60/	.	.	.	.	.
8ff	A3	/62/	2	3	9.89	H2	upt
.	.	/-62/	.	.	.	.	.
8dd-	Ab3	/70/	1	2	7.07	B2	ct
.	.	/-70/	.	.	.	.	.
8d-	C#4	/61/	6	3	7.135	Cis4	sus
.	.	/-61/	.	.	.	.	.
=2	=2	=2	=2	=2	=2	=2	=2
[4a-	r	.	5	1	r	r	.
.	.	.	7	3	5.5	Heses2	ct
4a-]	D4	/48/ /52/	1	2	8.11	C3	ct
.	.	/-48/	.	.	.	.	.
.	D4 F4	/-52/	2	3	7.33 6.4	C3 Es3	ct
=3	=3	=3	=3	=3	=3	=3	=3
r	G4	.	r	1	r	H2 D3	.
==	==	==	==	==	==	==	==
*-	*-	*-	*-	*-	*-	*-	*-
```

Executing the command

```
freq -tx input > output.frq
```

produces the following result:

```
!! 'freq' example.
**freq	**freq	**freq	**deg	**metpos	**freq	**freq	**embell
*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4
*	*	*	*	*tb8	*	*	*
=1	=1	=1	=1	=1	=1	=1	=1
622.25	415.30	261.63	1foo	1	r	103.83	ct
.	.	.	.	.	.	.	.
698.46	220.00	293.66	2	3	481.97	123.47	upt
.	.	.	.	.	.	.	.
554.37	207.65	466.16	1	2	273.21	116.54	ct
.	.	.	.	.	.	.	.
277.18	277.18	277.18	6	3	277.16	277.18	sus
.	.	.	.	.	.	.	.
=2	=2	=2	=2	=2	=2	=2	=2
415.30	r	.	5	1	r	r	.
.	.	.	7	3	187.76	110.00	ct
.	293.66	130.81 164.81	1	2	340.92	130.81	ct
.	.	.	.	.	.	.	.
.	293.66 349.23	.	2	3	289.24 234.47	130.81 155.56	ct
=3	=3	=3	=3	=3	=3	=3	=3
r	392.00	.	r	1	r	123.47 146.83	.
==	==	==	==	==	==	==	==
*-	*-	*-	*-	*-	*-	*-	*-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the **kern spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the **pitch, **MIDI, and **cocho spines have been stripped away (due to the **-x** option).

FILES

The file x_option.awk is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cbr** (2), **cbr** (4), ****cents** (2), **cents** (4), ****cocho** (2), **cocho** (4), ****freq** (2), ****fret** (2), ****kern** (2), **kern** (4), ****MIDI** (2), **midi** (4), ****pitch** (2), **pitch** (4), ****semits** (2), **semits** (4), ****solfg** (2), **solfg** (4), ****specC** (2) **specC** (4), ****Tonh** (2), **tonh** (4)

NAME

hint — determine harmonic intervals between concurrent pitches

SYNOPSIS

hint [**-a**|**-l**] [**-cdu**] [**-s** *regexp*] [*inputfile* ...] [> *outputfile.hnt*]

DESCRIPTION

The **hint** command outputs the harmonic interval distances between simultaneous notated pitches. Output pitch intervals are expressed as a combination of diatonic interval size and interval quality (such as a 'perfect fifth' or 'minor ninth'). Standard musical abbreviations are used (e.g., m2 - minor second; P4 - perfect fourth; A6 - augmented sixth; dd7 - doubly diminished seventh). By way of illustration, **hint** will change the `**pitch` diad *C4-E4* (or *E4-C4)* — to the interval token *M3*.

The **hint** command determines harmonic intervals for pitch tokens spanning all pitch-related spines in an input record; this includes multiple-stops within spines. In the default operation, a single output interpretation (`**hint`) is generated for any given input. All pitches in a given sonority (sounding moment) are first sorted in ascending pitch order. If the **-u** option is selected, duplicate pitches (unisons) are treated as a single pitch-instance when calculating the intervening intervals. Intervals are then calculated between successive pairs of pitches — ordered from low to high. A single data token is output — representing all intervals in a given sonority. If more than one interval is present, the interval tokens are assembled as Humdrum multiple-stops within the output data token.

Input records containing no pitch tokens result in the outputting of a null token (.). Input records containing only rests result in the outputting of a rest token (r). Input records containing only a single pitch result in the outputting of a hyphen (-). Input records containing only a single duplicated pitch result in the outputting of the interval (P1) — unless the **-u** option is selected, in which case the a hyphen is output (-).

The **hint** command recognizes and echoes "common-system" barlines (see *barlines (2)*). It is also able to handle multiple-stops.

By defining *regular expression* patterns, the user may select which data records should be ignored by **hint.** (See EXAMPLES below.)

Note that the output spine generated by **hint** preserves the same record-type structure as the input, and so may readily be pasted with the input file using the Humdrum **assemble** command.

The **hint** command is able to accept any of the pitch-related representations listed below. For descriptions of the various input representations refer to Section 2 *(Representation*

Reference) of this reference manual.

It is recommended that output files produced using the **hint** command should be given names with the distinguishing '.hnt' extension.

`**kern`	core absolute pitch representation
`**Tonh`	German pitch system
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**solfg`	French solfège system (fixed 'doh')

Input representations processed by **hint**.

OPTIONS

The **hint** command provides the following options:

-a	calculate all intervals by permuting all pitches present
-c	output compound intervals as non-compound intervals
-d	output diatonic interval size only, without the interval quality
-h	displays a help screen summarizing the command syntax
-l	calculate intervals with respect to the lowest pitch present
-s *regexp*	skip; completely ignore records matching *regexp;* (output null token)
-u	eliminate unisons from the output

The **-a** and **-l** options are mutually exclusive.

Options are specified in the command line.

The **-a** option causes all permuted intervals within a sonority to be output rather than only those intervals between successive pitch-ordered pitches. Hence, the sonority (E4 G4 C5) will produce the output m3 m6 P4 rather than m3 P4. A tandem interpretation (`*all`) is added to the output in order to indicate that the interval content is exhaustive. Note that the order of the intervals in the output token no longer has any significance when the **-a** option is invoked.

The **-c** option causes all compound intervals to be output as non-compound equivalents. Compound intervals are defined as those intervals greater than or equal to an octave in size. Hence, a major tenth interval will be output as a major third, and an octave will be output as a perfect unison.

The **-d** option causes only the diatonic interval size to be output. In this case, the interval quality signifiers (`AdmMP`) will be suppressed. For example, with the **-d** option, the output token '3' signifies any interval of a third, including major, minor, diminished, etc.

The **-l** option causes **hint** to calculate all intervals with respect to the lowest notated pitch present. Hence, the input sonority (G4 E4 C4) will produce the output M3 P5 rather than M3 m3.

The **-s** option allows the user to define a regular expression, that if matched, causes a null token to be output for the given record.

The **-u** option eliminates duplicate pitches in interval calculations. For example, rather than outputting P1 M3 P1 for the sonority (C4 C4 E4 E4), the **-u** option will result in the output M3. In the case of a sonority consisting of a repeated single pitch, the **-u** option will cause a hyphen (-) to be output.

EXAMPLES

The various aspects of the **hint** command are illustrated in the following examples. Consider the following input:

```
**kern    **kern    **pitch   **commentary
=1        =1        =1        barline
8c        8e        G4        C major triad
8g        8c        E4        reordered pitches
4r        4r        r         rest
.         .         .         null tokens
4C        4e        G5        open position triad
=2        =2        =2        barline
4C 4E     4G 4c     C5        multiple-stops
4CC 4r    4e 4g     r         mixed notes & rest
8C        8C        .         unison
8C        8r        .         single pitch
=3        =3        =3        barline
*-        *-        *-        *-
```

Using the default invocation, the **hint** command transforms the above input as follows:

```
**hint
=1
M3 m3
M3 m3
r
.
M10 m10
=2
M3 m3 P4 P8
M17 m3
P1
-
=3
*-
```

The **hint** command correctly echoes (and ignores) both rests and kern-like barlines — as illustrated in the first and fourth output data records. The second output data record (M3 m3) indicates that two intervals are present: the first interval is a major third above the lowest pitch, and the second interval is a minor third above the other pitch. The third data record shows that rearranging the order of the input pitches has no effect on the output.

Notice that the null-token in the sixth record has been echoed. Null-tokens have no effect on interval calculations and are treated as though they are non-existent. Multiple-stops are treated the same as if each pitch was in a separate spine (eighth data record), and rests within a sonority containing pitches are ignored (ninth data record). The perfect unison signifier (P1) is output only if more than one pitch is present (tenth data record). If a single pitch is present in the input, a hyphen is output rather than the P1 token (eleventh data record).

The **-d** option causes only the diatonic interval sizes to be output as follows:

```
**hint
=1
3 3
3 3
r
.
10 10
=2
3 3 4 8
17 3
1
-
=3
*-
```

The **-s** (skip) option can be used to allow the user to selectively identify records that should not be involved in processing. Consider the command `hint -s r input > output.hnt` will cause any data token containing the letter 'r' to suspend the calculation of any harmonic intervals for the current record. The corresponding `**hint` output for data records matching the pattern will consist simply of a null token. Given the sample input, intervals will be calculated only when none of the pitch-related spines contain a rest.

Given the first three spines of the above sample input (i.e. without the `**commentary` spine), the command

```
hint -cu -s r input > output
```

will produce the following output:

```
**hint
=1
M3 m3
M3 m3
r
.
M3 m3
=2
M3 m3 P4
.
-
-
=3
*-
```

Notice that the major and minor tenths in the sixth data record have been rendered as major and minor thirds. Also note that the perfect unison in the tenth data record has been output as a hyphen — and that the P1 formerly present in the eighth data record has disappeared. The presence of the rest in the original ninth data record has caused the outputting of a simple null token.

The **-l** option causes **hint** to calculate intervals with respect to the lowest pitch present in the sonority. For example, with the above sample input, the **-l** option would produce the following output:

```
**hint
*
=1
M3 P5
M3 P5
r
.
M10 P19
=2
M3 P5 P8 P15
M17 P19
P1
-
=3
*-
```

The **-a** option calculates all possible intervals by pairing all pitches present in a given sonority. The order of the output intervals conforms to the following standard: all pitches are sorted from low to high; intervals are determined as 1-2, 1-3, 1-4, etc., 2-3, 2-4, etc. By way of example, the following input:

```
**kern      **kern
4C 4E 4G    4c 4c 4r
*-          *-
```

would produce the following output:

```
**hint
*all
M3 P5 P8 P8 m3 m6 m6 P4 P4 P1
*-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****deg** (2), ****degree** (2), ****hint** (2), **humsed** (4), ****kern** (2), ****mint** (2), **mint** (4), **recode** (4), **regexp** (6), ****semits** (2), **semits** (4), **solfg** (2), ****Tonh** (2), **xdelta** (4), **ydelta** (4)

NAME

humdrum — general syntax checker for Humdrum files

SYNOPSIS

humdrum [-v] [*inputfile* ...]

DESCRIPTION

The **humdrum** command identifies whether an input stream conforms to the Humdrum representation syntax. Error messages are issued where appropriate. If the input conforms to the Humdrum syntax, then the **humdrum** command produces no output, unless the verbose (**-v**) option is invoked.

Formally, a Humdrum representation may be defined as any ASCII input containing zero or more *comments, data records* or *interpretations* — with the restriction that no *data records* or *local comments* appear without a prior *exclusive interpretation,* and that the file maintain a coherent *spine* organization.

Humdrum comments are records (lines) that begin with an exclamation mark (!). Local comments begin each active spine with a single exclamation mark, whereas gobal comments begin with two exclamation marks (!!) at the beginning of the record.

Humdrum interpretations are records that begin with an asterisk (*). Tandem interpretations begin each active spine with a single asterisk, whereas exclusive interpretations begin the record with one or more asterisks and have at least one active spine beginning with two asterisks. Spine-path indicators are special types of interpretations that include any one of the following tokens: *+ *- *v *^ or *x. Spine-path indicators cannot appear on the same line with a tandem or exclusive interpretation.

Any record that does not begin with either an exclamation mark of asterisk is a data record.

The following table defines some of the pertinent Humdrum terminology.

record	a line of text; a sequences of (zero or more) ASCII characters terminated by a carriage return or newline character
data record	any *record* which is not a *comment* or an *interpretation*
comment	a *global* or *local comment*
global comment	any *record* beginning with two exclamation marks (!!)
local comment	any *record* beginning with a single exclamation mark (!)
interpretation	an *exclusive* or *tandem interpretation*, or a *spine-path indicator*
exclusive interpretation	any *record* beginning with one or more asterisks (**) and having at least one active spine beginning with two asterisks
tandem interpretation	any *record* in which each active spine begins with a single asterisk, excluding spine-path indicators
spine-path indicator	one of the following five *interpretations*: *+ *- *v *^ *x
null token	the period (.) either alone on a single *record* or separated from other characters by tabs
spine	a column of information — including *data records, local comments*, and *interpretations*.

Some Humdrum Definitions

The following files may satisfy the Humdrum syntactical requirements:

1. a file containing data records preceded by at least one exclusive interpretation
2. a file containing only comments and interpretations
3. a file containing only interpretations
4. a file containing only global comments
5. a totally empty file.

Additional interpretations may be added throughout the file. Global comments may appear anywhere in the file. Local comments and data records can appear only after a spine has been initiated via an exclusive interpretation.

Illegal constructions in Humdrum include the following:

1. empty record (i.e. a record containing only a carriage return or newline)
2. record containing only tabs
3. any record beginning with a tab
4. any record ending with a tab (except in a global comment)
5. any record containing two successive tab characters (except in a global comment)
6. any data record with fewer or more spines than the immediately preceding data or exclusive interpretation record
7. spine-path record containing only one join-spine indication
8. spine-path record containing only one exchange-spine indication, or containing more than two exchange-spine indicators.

SPINE ORGANIZATION

A Humdrum file must maintain a coherent spine organization. Spines are columns of information separated by tabs. Each spine must be labelled with an exclusive interpretation. If the number of spines varies in a given file, spines must be added, terminated, split, joined, or exchanged using the appropriate spine path indicators. Spines may (1) start, (2) terminate, (3) split (into two), (4) join (*n*-into-one), and (5) exchange.

*+	- start a new spine
*-	- terminate a current spine
*^	- split a spine (into two)
*v	- join (two or more) spines into one
*x	- exchange the position of two spines

Spine-path Indicators

The use of various spine-path indicators is illustrated below:

```
1	2	3
*	*+	*
*	*	**inter	*		(must follow immediately after)
1	2	new	3

1	2	3
*	*-	*
1	3

1	2	3
*	*^	*
1	2a	2b	3

1	2	3
*	*v	*v
1	2&3

1	2	3
*x	*x	*
2	1	3
```

More complex examples:

```
1	2	3	4	5
*	*-	*	*-	*
*v	*v	*v
1&3&5
```

```
1       2     3     4
*       *     *v    *v
*       *x    *x
*v      *v    *
1&3&4   2

1       2     3     4       4b
*v      *v    *^    *^
1&2     3a    3b    4a      4b

1       2     3     4       5
*       *-    *     *       *
*v      *v    *     *
*       *^    *
*       *     *+    *
*       *     *     **inter *
1&3     4a    4b    new     5
```

OPTIONS

The **humdrum** command provides the following options:

-h displays a help screen summarizing the command syntax
-v verbose mode

Options are specified in the command line.

The **-v** option invokes the verbose mode which provides summary information and statistics concerning the input file. A list of all of interpretations found in the file is output. In addition, an inventory of all of the ASCII signifiers (characters) found in the data records is output. The total number of data records is also provided, as well as the number of null tokens encountered.

The verbose summary also produces a number of statistics related to the spine-organization in the Humdrum file. Both the minimum and maximum number of spines are identified. The paths of the various spines through the file are also summarized. Specifically, **humdrum** indicates the number of terminated spines, the number of new spines introduced, the number of joined spines, the number of split spines, and the number of exchanged spines.

EXAMPLES

The following is a sample input that conforms to the Humdrum syntax.

```
!! This is a global comment.
!! The following line specifies three
!! interpretations called "inter":
**inter       **inter       **inter
!! The next line has three local comments.
! flute       ! oboe        ! fingering
We            are           ASCII
data          tokens        .
We are        data as       well.
!! The above three lines are data records.
!! More examples of data records:
76.3          x+y           L(-)%4^
___           __            _____
>             <             |||
!! The next data record contains
!! only null tokens:
.             .             .
!! Some spine-path exchanges:
!Spine1       !Spine2       !Spine3
*x            *x            *
*             *x            *x
!Spine2       !Spine3       !Spine1
Monday        Tuesday       Wednesday
4:00          5:00          6:00PM
!! Some null comments follow:
!!
!             !             !
!! New exclusive interpretations:
**foo         **bar         **ding
!! A tandem interpretation in spine #2:
*             *bop          *
More          data          tokens.
!! Some spine-path terminators:
*-            *-            *-
```

Where the verbose (**-v**) flag has been selected, the corresponding output for the above file would be:

```
Inventory of Interpretations:                          HUMDRUM FILE SUMMARY

        Interpretations  Occurrences
        **inter          3
        **foo            1
        **bar            1
        **ding           1
        *                2
        *bop             1

Inventory of data token signifiers:     %()+-.034567:<>
                                        ACILMPSTW^_
                                        adeklnorstuwxy

Number of global comments:      14      Number of local comments:       4
Maximum # of concurrent spines: 3       Minimum # of concurrent spines: 3
Number of data records:         10      Number of null tokens:          4


Changes in number of spines:
        New spines:        0
        Terminated spines: 3
        Split spines:      0
        Joined spines:     0
        Exchanges spines:  4
```

FILES

The **humdrum** command uses a kornshell script to invoke an executable. In order to avoid conflict, the executable file is named `humdrum_.exe` (`humdrum_` on UNIX) rather than `humdrum.exe`.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

census (4), **proof** (4)

DIAGNOSTICS

The following list tabulates all of the potential errors and warnings issued by the **humdrum** command.

ERROR 1:	Record containing add-spine indicator has not been followed by exclusive interpretation for that spine in line
ERROR 2:	Illegal empty record in line
ERROR 3:	Leading tab in line
ERROR 4:	Trailing tab in line
ERROR 5:	Consecutive tabs in line
ERROR 6:	Missing initial asterisk in interpretation keyword in line
ERROR 7:	Null exclusive interpretation found in line
ERROR 8:	Incorrect number of spines in interpretation record in line
ERROR 9:	Local comment precedes first exclusive interpretation record in line
ERROR 10:	Number of sub-comments in local comment does not match the the number of currently active spines in line
ERROR 11:	Missing initial exclamation mark in local comment in line
ERROR 12:	Data record appears before first exclusive interpretation record in line
ERROR 13:	Number of tokens in data record does not match the number of currently active spines in line
ERROR 14:	All spines have not been properly terminated in file
ERROR 15:	First exclusive interpretation record contains a null interpretation in line
ERROR 16:	First exclusive interpretation record contains a spine-path indicator in line
ERROR 17:	First exclusive interpretation record contains a non-exclusive interpretation in line
ERROR 18:	Spine-path indicators mixed with keyword interpretations in line
ERROR 19:	Improper number of exchange-path indicators in line
ERROR 20:	Single join-path indicator found at end of interpretation record in line
ERROR 21:	Join-path indicator is not adjacent to another join-path indicator in line
ERROR 22:	Exclusive interpretations do not match for designated join spines in line
ERROR 23:	Leading spaces in token in line
ERROR 24:	Trailing spaces in token in line
ERROR 25:	Consecutive spaces in token in line
ERROR 26:	Multiple-stop contains null token in line
WARNING 1:	Local comment may be mistaken for global comment in line
WARNING 2:	Data token may be mistaken for global comment in line
WARNING 3:	Data token may be mistaken for local comment in line
WARNING 4:	Data token may be mistaken for exclusive interpretation in line
WARNING 5:	Data token may be mistaken for tandem interpretation in line

Potential errors and warnings issued by **humdrum.**

NAME

humsed — stream editor for Humdrum inputs

SYNOPSIS

humsed [**-E**] '*sed_instruction[;sed_instruction;...]*' [*inputfile* ...] [> *outputfile*]
humsed [**-E**] [**-f** *scriptfile*] [*inputfile* ...] [> *outputfile*]

DESCRIPTION

The **humsed** command provides a stream-editor for Humdrum data records. A stream-editor is a non-interactive editor that automatically processes a given input according to a user-specified set of editing instructions. Possible editing operations include substitution, deletion, transliteration, file-read, and file-write. By default the output is sent to the display, however the output can be routed to a file using file redirection (> *outputfile).*

The **humsed** command is fashioned after the UNIX **sed** stream editor. In contrast to **sed, humsed** editing instructions are applied only to Humdrum data records; Humdrum *interpretations* and *comments* are not affected by **humsed.**

The **humsed** command accepts one or more **sed** instructions. Instructions are specified on the command-line within a pair of single quotes. Where more than one editing instruction is specified, successive instructions are separated by a semicolon. Alternatively, instructions may be executed from a *scriptfile* using the **-f** option. If instructions are provided both on the command-line as well as via a *scriptfile,* the command line instructions are performed prior to the *scriptfile* instructions.

Permissible instructions include **s** for substitution, **y** for transliteration, **d** for deletion, **i** for insertion, **a** for append, **r** for file-read, and **w** for file-write. Each instruction may be preceded by an optional *regular expression* that limits the scope of the editing instruction only to those data records matching the regular expression. For example, the user may replace all occurrences of 'X' with 'Y' — provided the signifier 'Z' also occurs in the same data record. In the case of the delete **(d)** instruction, failing to specify a preceding regular expression will result in the deletion of all data records in the input.

For further information concerning the syntax and use of **humsed** editing instructions, refer to the documentation for the UNIX **sed** command.

OPTIONS

The **humsed** command provides the following options:

-h	displays a help screen summarizing the command syntax
-E	invoke Extended Regular Expression syntax
-f *scriptfile*	execute editing instructions from the file *scriptfile*

Options are specified in the command line.

With the **-E** option, **humsed** invokes the "extended" regular expression syntax, rather than the normal or "basic" regular expression syntax. With extended regular expressions, the following additional operations are supported: one-or-more (+), zero-or-one (?), logical OR (|), precedence grouping ( ), and alphanumeric token start and end anchors < >.

Note that not all systems support extended regular expressions for the **sed** command; on such systems the **-E** option for **humsed** is ineffective and may result in an error.

The **-f** options allows the user to specify a *scriptfile* that contains a set of editing instructions. Instructions in *scriptfile* are executed after any command-line editing scripts.

EXAMPLES

The following examples illustrate the substitution, transliteration, deletion, file-read and file-write instruction provided by **humsed.**

Simple substitution:

```
humsed 's/A/X/g' ragtime
```

The above command replaces the upper-case letter A by the upper-case letter X. Without the `g` (global) modifier, only the first occurrence of an "A" in each data record would be modified. The use of `g` applies the substitution instruction to *all* occurrences in a data record.

Substitution commands can be preceded by another regular expression that limits the selection of records that are affected by the substitution. For example, the following command eliminates all measure numbers in a `**kern` representation:

```
humsed '/=/s/[0-9]*//g' jellyroll
```

Rather than simply eliminating all numerical data, the initial regular expression `(/=/)` limits the substitution operation to those data records contain the `**kern` barline signifier `(=)`.

More complicated substitutions may involve compound (two or more) instructions. Instructions are separated by a semicolon, and are executed in succession for each data record. Consider the following command:

```
humsed 's/4[A-G]/8&/g;s/84/8/g' chicago > fastbass
```

This command changes all quarter-note pitches (in a `**kern` representation) below middle

C to eighth-note durations, while leaving quarter-notes above middle C unchanged. The first substitution instruction `(s/4[A-G]/8&/g)` searches for all strings beginning with the number 4, followed by one of the upper-case letters A to G. It then prepends the number 8; thus the token 4F will be replaced by 84F. (Note that the ampersand (&) in the substitution denotes the matched string found by the target regular expression.) The second substitution `(s/84/8/g)` replaces the string 84 by the string 8. In short, tokens such as 4F and 4CC# will be modified to 8F and 8CC# respectively — whereas tokens such as 2F and 4cc# will remain unmodified.†

The transliteration instruction (**y**) provides a short-cut for multiple single-character substitutions. For example, the following command replaces A with 0, B with 1, C with 2, etc. for the letters A to J:

```
humsed 'y/ABCDEFGHIJ/0123456789/' dixieland
```

Substitutions are organized by mapping each element in the first character string with the corresponding element in the second string. The first and second character strings must contain the same number of characters.

The delete instruction is preceded by a regular expression, followed by the single letter **d.** The following command deletes all data records containing the lower-case letter "r".

```
humsed '/r/d' swing
```

The file-write instruction (**w**) provides a way of copying selected material to a specified output file. Consider the following command:

```
humsed '/;/w pauses' bigband
```

This command identifies all data records in the file "bigband" that contain a semicolon (the `**kern` pause signifier) and copies them into the file "pauses." Recall that **humsed** operates only on Humdrum data records, so the **w** command will cause only data records to be outputted. Hence the resulting file "pauses" will not be a valid Humdrum file. (If the user wishes the extracted material to be in a valid Humdrum format, this could be done using the Humdrum **yank** command:`yank -m ';' 0 bigband > pauses.`)

The **humsed** command can also be used to read (**r**) material from a specified file whenever a certain condition occurs in the input stream. For example, the following command could be used to search for `**kern` pause signifiers (;) and add a global comment indicating the presence of a pause.

```
humsed '/;/r comment' bebop
```

† Note that this command is inadequate if 24th notes (thirty-second note triplets) are present in the input — since they will be transformed to 28th notes.

— where the file "comment" contains the following global comment:

```
!! A pause.
```

PORTABILITY

Any system supporting the UNIX-style **sed** command. Note that the **-E** option is a non-POSIX extension currently supported only by the MKS toolkit. It is hoped that in the future, other systems will support extended regular expression syntax for **sed.**

SEE ALSO

awk (UNIX), **regexp** (4), **regexp** (6), **rid** (4), **sed** (UNIX), **vi** (UNIX), **yank** (4)

WARNINGS

In the process of modifying text, it is possible to transform inadvertently Humdrum data records into interpretation records or comments. Particular caution should be exercised when inserting exclamation marks or asterisks.

In addition, it is possible to disrupt the spine structure by inserting or deleting tabs. Substitutions may result in empty lines or extra spaces that render the file no longer consistent with the Humdrum syntax.

LIMITS

Note that the extended regular expression mode (**-E**) is not available on most UNIX systems.

NAME

humver — display Humdrum toolkit version and copyright information

SYNOPSIS

humver [-s]

DESCRIPTION

The **humver** command tells the user which version or release of the Humdrum Toolkit is installed or available on a given computer system. The command also prints copyright information pertaining to the Humdrum Toolkit.

OPTIONS

The -s option suppresses output and returns a value to the shell corresponding to the Humdrum Toolkit version number. (This option is useful in allowing shell programs to determine the version number without sending output to the standard output.)

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

NAME

infot — calculate information theory measures

SYNOPSIS

infot -b [**-H**] [**-x** *regexp*] [*inputfile* ...]
infot -n [**-H**] [**-x** *regexp*] [*inputfile* ...]
infot -p [**-H**] [**-x** *regexp*] [*inputfile* ...]
infot -s [**-x** *regexp*] [*inputfile* ...]

DESCRIPTION

The **infot** command provides a general-purpose tool for measuring the probability relationships between user-selected data tokens. Given a specified input stream, **infot** can calculate one of several pertinent information-theoretic values. The values may be calculated with reference to an independent repertoire, or may be calculated as so-called "self-information."

In conjunction with other Humdrum tools (notably the **context** and **humsed** commands), **infot** permits sophisticated information-theoretic analyses to be carried out, including calculations of information flow, short-term conditional probabilities, and longer-term *m-dependency* analyses. Alternatively, a simple set of summary statistics can be requested. In most cases, users will want to use **infot** to generate outputs that are suitable for further processing.

Input to **infot** is restricted to a single spine. However, the input data tokens may contain multiple-stops representing complex contextual information (such as produced by the **context** command).

For the entire input, **infot** tabulates the total number of occurrences of each unique data record (hereafter referred to as 'states'). For the **-n, -p** and **-b** options, **infot** outputs a two-column list where the left column indentifies each unique state and the right column provides one of several corresponding calculated measures. With the **-n** option, this measure is merely an integer count of the number of occurrences of each corresponding state. With the **-p** option, this measure is a probability of occurrence for each state. With the **-b** option, this measure identifies the information content for the corresponding state in bits.

Information content (H) in bits is calculated according to the classic equation devised by Shannon and Weaver (see REFERENCES):

$$H = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i}$$

where H is the average information (in bits), N is the number of possible unique states in the repertoire, and p_i is the probability of occurrence of state i from the repertoire.

Note that the outputs produced by **infot** do not conform to the Humdrum syntax.

OPTIONS

The **infot** command provides the following options:

-b	output information (in bits) for each unique data token
-h	displays a help screen summarizing the command syntax
-H	format output as **humsed** commands
-n	output frequency count for each unique data token
-p	output probability value for each unique data token
-s	output information-related summary statistics
-x *regexp*	exclude tokens matching *regexp* from calculations

Options are specified in the command line.

With the **-n** option, **infot** outputs a two-column list where the left column indentifies each unique state present in the input, and the right column provides an integer *count* indicating the number of occurrences for the corresponding state.

With the **-p** option, **infot** outputs a two-column list where *probabilities* of occurrence are output in the right-hand column, rather than counts.

With the **-b** option, **infot** outputs the information (in bits) as calculated according to the Shannon-Weaver equation.

EXAMPLES

The use of **infot** is illustrated in the following examples. Consider the following input:

```
**foo
A
B2
C-c
A
B2
A
A
B2
C-c
A
A
X Y
*-
```

A simple command invocation would use the **-n** option to count the number of occurrences of each unique data token (or state):

```
infot -n input
```

The corresponding output is:

```
A     6
B2    3
C-c   2
X Y   1
```

The tallies indicate that state 'A' occurs 6 times, and that the least common state ('X Y') occurs just once. If we had invoked the **-p** option, the counts would be replaced by probabilities. The command:

```
infot -p input
```

produces the following output:

```
A     0.500
B2    0.250
C-c   0.167
X Y   0.083
```

Alternatively, the **-b** option:

```
infot -b input
```

would output information measures for each state, in bits:

```
A     1.000
B2    2.000
C-c   2.585
X Y   3.585
```

In the case of the **-s** option, summary statistics would be output, rather than a two-column list. For the above input, the following summary statistics would be generated:

```
Total number of input states in message:   4
Total information of message  (in bits):   20.7549
Total possible  information for message:   24
Info  per state  for equi-prob  distrib:   2
Average information conveyed  per state:   1.72957
Percent redundancy  evident  in message:   13.5213
```

The first line of output merely indicates the number of unique states found in the input (in this case just 4). The fifth output line indicates the average information conveyed per state (in bits). The fourth output line indicates the theoretical maximum average information per state that could be communicated by a system having four states. The third line indicates the maximum possible information that could be communicated in a message of the same length as the input — given the theoretical maximum average information. (Since there are 12 data records, this value is simply 12×2 bits, or 24 bits.) The second output line gives the actual total information for the given input message. (This is always less-than, or equal-to the maximum theoretical value.) The final line indicates the amount of redundancy — as a percentage. That is, this value contrasts the actual information conveyed with the theoretical maximum.

In general, note that as the probabilities of the input states approach equivalence, the redundancy approaches zero and the average information content approaches the theoretical maximum.

Consider now an example where a large number of messages from a repertoire (dubbed `repertoire`) is passed to **infot**:

```
infot -b repertoire
```

Suppose that the following output is produced:

```
ABC     3.124
BAC     1.306
C C D   1.950
X       5.075
XYZ     19.334
```

This result indicates that, although there might have been hundreds of data tokens processed in the repertoire, only five different unique states were present. The greatest information content (lowest probability) is associated with the state `XYZ` (19.334 bits), whereas the lowest information content (highest probability) is associated with the state `BAC` (1.306 bits). Notice that the multiple-stop `C C D` is treated as a single state.

Now imagine we had another message presumed to belong to the same repertoire as our input. We would like to trace how the information increases and decreases over the course of this new 'message'. This goal involves a two-part operation. First, we re-invoke **infot** adding the **-H** option, and redirect the output to a file `replace`:

```
infot -bH repertoire > replace
```

This causes **infot** to produce as output a set of **humsed** commands. Given the identical `repertoire` input, the following output is sent to the file `replace`:

```
s/^ABC$/3.124/g; s/^ABC    /3.124/g; s/    ABC$/3.124/g; s/  ABC      /3.124/g
s/^BAC$/1.306/g; s/^BAC    /1.306/g; s/    BAC$/1.306/g; s/  BAC      /1.306/g
s/^C C D$/1.95/g; s/^C C D /1.95/g; s/     C C D$/1.95/g; s/ C C D    /1.95/g
s/^X$/5.075/g; s/^X        /5.075/g; s/    X$/5.075/g; s/    X        /5.075/g
s/^XYZ$/19.334/g; s/^XYZ  /19.334/g; s/    XYZ$/19.334/g; s/ XYZ      /19.334/g
```

Although these commands may appear somewhat cryptic, they merely instruct the Humdrum stream editor (**humsed**) to replace all occurrences of the five data tokens (in any input file) by the corresponding numerical values — in this case, values that represent the number of bits of information.

The following file (called `input`) contains the message of interest:

```
**bar
BAC
BAC
C C D
.
=
*
C C D
XYZ
X
ABC
BAC
*-
```

This file can be transformed so that the data tokens are replaced by corresponding information values as determined from the original repertoire. This is done by invoking the **humsed** command, and providing it with the substitution commands held in the file `replace`:

```
humsed -f replace input > output
```

The resulting output file would be as follows:

```
**bar
1.306
1.306
1.950
.
=
*
1.950
19.334
5.075
3.124
1.306
*-
```

Note that data tokens in message that do not appear in the probability list (such as the equals-signs) remain unmodified.

Several interpretations may be made about this message. For example, the above passage appears to show a pattern of initially low information that increases and then decreases toward the end of the passage. This suggests that the beginning and ending of this passage are more highly constrained or stereotypic than the middle part of the passage.

Summing together the individual information values for this passage, the total message conveys 35.35 bits. For five states, the maximum average information is 2.322 bits per state, and so the expected maximum for a message consisting of 8 items would be 8 × 2.322 or 18.58 bits. This suggests that this message is considerably less banal, (less redundant or more unique) than a typical message from the original repertoire. In particular, the occurrence of the state 'XYZ' has a low probability of occurrence — and is likely to be a distinctive feature of this passage.

In the above examples, only simple (zeroth-order) probabilities have been examined. Higher-order and *m*-dependency probabilities may be measured by reformulating the input using the **context** command.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

context (4), **humsed** (4), **patt** (4), **pattern** (4), **simil** (4)

REFERENCES

Abraham Moles, *Information Theory and Esthetic Perception,* Urbana: University of Illinois Press, 1968.

Shannon, C. E., & Weaver, W. *The Mathematical Theory of Communication.* Urbana: University of Illinois Press, 1949.

Wong, A. K. C., & Ghahraman, D. A statistical analysis of interdependence in character sequences. *Information Sciences,* Vol. 8 (1975) pp. 173-188.

NAME

iv — determine interval vectors for successive vertical sonorities

SYNOPSIS

iv [*inputfile* ...] [> *outputfile.iv*]

DESCRIPTION

The **iv** command is used to determine the *interval vector* for any of five set-theory related inputs: pitch (`**semits`), pitch-class (`**pc`), normal form (`**nf`), prime form (`**pf`), or Fortean set name (`**pcset`). An interval vector is a six-element numerical list that indicates the abundance of various interval-classes (from 1 semitone to 6 semitones) for some pitch-class set. See REFERENCES below.

When provided with `**semits` or `**pc` inputs, **iv** treats each input record as a set of pitches. Unisons and other pitch-class duplications have no effect on the output. Rests within a set of pitches are ignored; where an input record consists solely of one or more rests, a null-token is output.

The **iv** command is able to translate any of the representations listed below. For descriptions of the various input representations (including `**iv`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **iv** command should be given names with the distinguishing '.iv' extension.

`**iv`	interval vector representation
`**nf`	normal form representation
`**pc`	pitch-class representation
`**pcset`	Fortean pitch-class set name
`**pf`	prime form representation
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)

Input representations processed by **iv**.

OPTIONS

The **iv** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

The following command outputs the interval vectors for successive sonorities in the input file `opus24`. The input may be pitches, pitch-classes, normal forms, Fortean set names, etc.

```
iv opus24 > opus24.iv
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

context (4), ****iv** (2), ****nf** (2), **nf** (4), ****pc** (2), **pc** (4), ****pcset** (2), **pcset** (4), ****pf** (2), **pf** (4), ****semits** (2), **semits** (4)

NOTE

The **iv** command is a shell script that invokes `pcset -v`.

REFERENCES

Allen Forte, *The Structure of Atonal Music.* New Haven: Yale University Press, 1973.

John Rahn, *Basic Atonal Theory.* New York: Longman Inc., 1980.

Straus, J. *Introduction to Post-Tonal Theory.* Englewood Cliffs, N.J.: Prentice Hall, 1990.

NAME

kern — translate pitch-related representations to `**kern`

SYNOPSIS

kern [**-x**] [*inputfile* ...] [> *outputfile.krn*]

DESCRIPTION

The **kern** command transforms various pitch-related inputs to corresponding `**kern` representations. For example, the `**pitch` token '`Ab2`' will be output as the `**kern` token '`AA-`'. Continuous pitch-related representations, such as frequency (`**freq`) and cents (`**cents`) are rounded-off to the nearest equally-tempered pitch. Hence, `**freq` values between 254.178 and 269.291 will be output as the `**kern` token for middle C — '`c`'.

Pitches in `**kern` are encoded as equally-tempered values at concert pitch. Kern is not able to represent pitch deviations from equal temperament. Diatonic pitch names are encoded using the letters A to G. Octaves are indicated by a system of upper- and lower-case letters, and by letter repetition. Middle C is represented by the single lower-case letter 'c'. The C an octave above is represented by two lower-case letters: 'cc' — with each successive octave adding another letter. The C an octave below middle C is represented by a single upper-case 'C'. The C an octave lower yet is represented by two upper-case letters: 'CC' — and so forth. Changes of octave are deemed to occur between the pitches B and C. Thus the B below middle 'c' is rendered as a single upper-case 'B'; the B below 'cc' is 'b' and so forth. Sharps are indicated by the octothorpe sign (#) whereas flats are indicated by the minus sign (–).

The **kern** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**kern`) refer to Section 2 *(Representation Reference)* of this reference manual.

`**cents`	hundredths of a semitone with respect to middle C=0
`**degree`	scale degree including octave designation
`**freq`	frequency in hertz (abbreviated Hz.)
`**fret`	fretted-instrument pitch tablature
`**MIDI`	Music Instrument Digital Interface key-press tablature
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **kern**.

For numerically-oriented inputs, such as `**cents`, `**freq`, `**MIDI`, `**semits`, and `**specC`, variant enharmonic spellings are selected for output according to the prevailing key signature or explicit key indication. (Refer to **key** and **key signatures** in Section 3 *(Humdrum Tandem Interpretations).* Hence, in the key of G minor, F-sharp and E-flat spellings will be output rather than G-flat and D-sharp. Kern recognizes the presence of key, or key signature tandem interpretations. If no key or key signature is encountered in the input, a default key of C major is assumed. Kern is sensitive to both *pitch-class* and *pitch-height* key signatures. In the case of pitch-height key signatures, the user can specify complex spelling preferences, such as F#2 rather than Gb2, but Gb3 rather than F#3, etc. See SAMPLE OUTPUT below.

It is recommended that output files produced using the **kern** command should be given names with the distinguishing '.krn' extension.

OPTIONS

The **kern** command provides the following options:

-h displays a help screen summarizing the command syntax
-x suppresses printing of non-pitch-related signifiers

Options are specified in the command line.

In the default operation, **kern** outputs any non-pitch-related signifiers in addition to the kern value. For example, the `**pitch` token "A6zzz" will result in the output "aaazzz" — that is, after translating A6 to "aaa", the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are upper- or lower-case letters from A-G. Consider the case of the `**freq` token "aA#5"; after translating "A#5" to "aa", the leading non-pitch-related signifier "a" will be prepended to the output, hence the value "aaa" — which will undoubtedly cause confusion. The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most inputs, the **-x** option is recommended.

SAMPLE OUTPUTS

The following example illustrates the use of **kern.** The input contains six pitch-related spines — two of which (`**deg` and `**cocho`) cannot be processed by **kern.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'kern' example #1
**specC  **pitch  **MIDI      **deg  **metpos  **cocho    **Tonh   **embell
*M2/4    *M2/4    *M2/4       *M2/4  *M2/4     *M2/4      *M2/4    *M2/4
*        *        *           *      *tb8      *          *        *
=1       =1       =1          =1     =1        =1         =1       =1
foo2000  G#4foo   /60/bar     1foo   1         r          Gis2     ct
.        .        /-60/       .      .         .          .        .
2321     A3+20    /62/        2      3         9.89       H2       upt
.        .        48/-62/     .      .         .          .        .
1807     Ab3      0/70/64     1      2         7.07       B2       ct
.        .        48/-70/     .      .         .          .        .
2487     C#4      /61/        6      3         7.135      Cis4     sus
.        .        /-61/       .      .         .          .        .
=2       =2       =2          =2     =2        =2         =2       =2
3323     r        .           5      1         r          r        .
.        .        .           7      3         5.5        Heses2   ct
3471     D4-8     /48/ /52/   1      2         8.11       C3       ct
.        .        /-48/       .      .         .          .        .
.        D4 F4    /-52/       2      3         7.33 6.4   C3 Es3   ct
=3       =3       =3          =3     =3        =3         =3       =3
r        G4       .           r      1         r          H2 D3    .
==       ==       ==          ==     ==        ==         ==       ==
*-       *-       *-          *-     *-        *-         *-       *-
```

Executing the command

```
kern -x input > output.krn
```

produces the following result:

```
!! 'kern' example #1
**kern   **kern   **kern   **deg    **metpos   **cocho    **kern   **embell
*M2/4    *M2/4    *M2/4    *M2/4    *M2/4      *M2/4      *M2/4    *M2/4
*        *        *        *        *tb8       *          *        *
=1       =1       =1       =1       =1         =1         =1       =1
bbb      g#       c        1foo     1          r          GG#      ct
.        .        .        .        .          .          .        .
dddd     A        d        2        3          9.89       BB       upt
.        .        .        .        .          .          .        .
aaa      A-       b-       1        2          7.07       BB-      ct
.        .        .        .        .          .          .        .
eeee-    c#       d-       6        3          7.135      c#       sus
.        .        .        .        .          .          .        .
=2       =2       =2       =2       =2         =2         =2       =2
aaaa-    r        .        5        1          r          r        .
.        .        .        7        3          5.5        BB--     ct
aaaa     d        C E      1        2          8.11       C        ct
.        .        .        .        .          .          .        .
.        d f      .        2        3          7.33 6.4   C E-     ct
=3       =3       =3       =3       =3         =3         =3       =3
r        g        .        r        1          r          BB D     .
==       ==       ==       ==       ==         ==         ==       ==
*-       *-       *-       *-       *-         *-         *-       *-
```

Both processed and unprocessed spines are output. Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**specC`, `**pitch`, and `**MIDI`, spines have been stripped away (due to the **-x** option).

Key signature sensitivity is illustrated in the following example. The input contains a "pitch-height key signature" — where flats and sharps pertain to only a specific absolute pitch. For example, Bb3 is preferred to A#3, although A#4 is preferred to Bb4. Similarly, C#4 is preferred to Db4, although Db5 is preferred to C#5.

```
!! 'kern' example #2
**semits
*K[B3-C4#F4#A4#D5-]
-2
10
1
6
13
*-
```

Notice in the corresponding output given below, that all pitches are rendered with the correct enharmonic spelling.

```
!! 'kern' example #2
**kern
*K[B3-C4#F4#A4#D5-]
B-
a#
c#
f#
dd-
*-
```

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), **cents** (4), ****degree** (2), **degree** (4), ****freq** (2), **freq** (4), ****fret** (2), **hint** (4), ****kern** (2), **mint** (4), ****MIDI** (2), **midi** (4), ****pitch** (2), **pitch** (4), **proof** (4), ****semits** (2), **semits** (4), ****solfg** (2), **solfg** (4), ****specC** (2), **specc** (4), ****Tonh** (2), **tonh** (4)

BUGS

When translating `**pitch`, `**solfg`, or `**Tonh`, inputs, **kern** ignores cents deviation. Hence C#6+80 is rendered as 'ccc#' rather than the nearest pitch 'ddd'.

WARNINGS

Humdrum representations are expected to avoid context dependency insofar as possible. This can lead to unexpected results. For example, the letter 'x' in `**pitch` is intended to signify the presence of a double sharp. Thus the `**pitch` input token 'xyzC4' is correctly translated by **kern** as 'yzc##'. Similarly, the `**pitch` input token '1yzC4' becomes 'yzCCC4'. (The first numerical value is interpreted as the octave number and the trailing number 4 is interpreted as a non-pitch-related signifier.)

NAME

key — estimate the key (tonic and mode) of a passage

SYNOPSIS

key [**-af**] [*inputfile* ...]

DESCRIPTION

The **key** command estimates the key of a given musical passage using Krumhansl's tonal hiearchy method. The command is restricted to identifying only those keys within the common major/minor tonal system. Modes outside the major/minor system are not recognized.

The input may be either `**semits` or `**kern` representations. The program adapts to input having varying numbers spines each with a different interpretation.

The non-optional output consists of three items of information:

1. the *estimated key* for the passage,
2. a *coefficient of correlation* (Pearson's "r") that measures how well the pitch organization of the musical passage conforms to an idealized major or minor key template, and
3. a *confidence score* that indicates how distinctive the key-match is compared with other competing keys. A confidence score of 100% indicates a very strong confidence in the key estimate; low confidence scores indicate that there is at least one other key that is a good alternative candidate.

The algorithm is based on Krumhansl's perceptually-based key-finding method (see reference). This method compares a given pitch-class frequency profile with two perceptually-determined prototypes (one each for major and minor modes). The coefficients used for these prototypes are those determined by Krumhansl & Kessler (1982). In order for the algorithm to work properly, durational information ought to be provided. When using `**semits` format input, best results are achieved when the input has a *time-base* format. (See the **timebase (4)** command.)

The **key** command is poor at distinguishing less common enharmonic keys. For example, it is unable to distinguish the following enharmonic spellings for tonic pitches: C-flat, B-sharp, E-sharp, F-flat. Also, **key** is unable to distinguish enharmonic spellings involving double- or triple- sharps or flats. That is, G double-sharp major is identified as A major. KEY is able to distinguish the more common enharmonic spellings (such as E-flat versus D-sharp).

There is no special output file-type designation.

OPTIONS

The **key** command provides the following options:

-a output correlation values for all keys
-f output frequencies for all pitch-classes
-h displays a help screen summarizing the command syntax

Options are specified in the command line.

The **-a** option will show all of the correlation coefficients for all 24 of the (enharmonic) major and minor keys.

The **-f** option will output the relative frequencies for each of the twelve chromatic pitch classes (in quarter-note durations).

SAMPLE OUTPUT

Without any options, typical outputs are of the following form:

```
Estimated key: B minor    (r=0.8442)     confidence: 51.3%
```

With both the **-f** and **-a** options invoked, a typical output is given below. The **-f** option causes 12 pitch-class tallies to be outputted. These values are given in acculumated whole-note durations. For example, the output: "PC[5]: 4.25" means that the enharmonic pitch-class "F" appears in the passage for the equivalent of 4 whole-notes plus a quarter-note duration. If inputs do not include durational information (such as in `**semits` input), each note is assigned the nominal duration of a quarter-note.

```
PC[0]: 5.50617
PC[1]: 0.375
PC[2]: 6.1875
PC[3]: 0
PC[4]: 5.625
PC[5]: 4.25
PC[6]: 1.25
PC[7]: 5.6875
PC[8]: 0.5
PC[9]: 4.625
PC[10]: 0.625
PC[11]: 4.40625
Tonic[0]     major 0.791744     minor 0.0962456
Tonic[1]     major -0.747033    minor -0.337397
Tonic[2]     major 0.506935     minor 0.535771
Tonic[3]     major -0.404982    minor -0.720203
Tonic[4]     major 0.0308014    minor 0.64007
Tonic[5]     major 0.475928     minor -0.13113
Tonic[6]     major -0.735928    minor -0.157988
Tonic[7]     major 0.772586     minor 0.205276
Tonic[8]     major -0.574103    minor -0.487743
Tonic[9]     major 0.232566     minor 0.66303
Tonic[10]    major -0.014411    minor -0.625767
Tonic[11]    major -0.334105    minor 0.319835
Estimated key: C major     (r=0.7917)     confidence:   5.7%
```

The **-a** option causes the tonic major and minor correlations to be printed for each pitch-class. Good key matches have a high positive correlation; the maximum value is 1.0.

In the above sample output, notice that the confidence score for the predicted key of C major (Tonic[0]) is quite low. The reason for this is that the correlation coefficient for A minor (Tonic[9]) is rather close to that for C major (0.791744 versus 0.66303) Note that confidence scores may be used as a simple index for estimating the tonal ambiguity or degree of chromaticism for a passage.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****kern** (2), **kern** (4), ****semits** (2), **semits** (4), **timebase** (4)

WARNINGS

As noted, **key** is very limited. When used to find "the key" of a particular work, it frequently produces incorrect results. In particular, for much tonal music, **key** has a tendency to mistakenly identify the dominant of the key rather than the tonic. See also the discussion under "proposed modifications."

BUGS

The current version does not handle multiple stops in `**kern` spines.

NOTES

Currently the program is sensitive to `**semits` and `**kern` interpretations. If no recognizable interpretation is given in the input stream, **key** assumes `**kern` compatible input. This may lead to erroneous results.

REFERENCES

Krumhansl, C. *Cognitive Foundations of Musical Pitch,* Oxford: Oxford University Press, 1990, Chapter 4.

Krumhansl, C. L. & Kessler, E. J. "Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys," *Psychological Review,* Vol. 89 (1982) pp.334-368.

PROPOSED MODIFICATIONS

The **key** command is likely to appeal to two types of uses: (1) those who are interested in knowing the key of a musical work without having to examine the work manually, or (2) those who are interested in a perceptual characterization of the tonality of a passage. These goals are quite different. The current **key** command implements the second approach; **key** is not very good at consistently and correctly identifying "The Key" of typical tonal works. Another command (or variant of the **key** command) might use contextual heuristics (such as looking at the final chord of a work, or examining cadences) in order to better identify "the key" of a work.

NAME

melac — calculate melodic accent values for successive pitches

SYNOPSIS

melac [*inputfile.sem* ...] [> *outputfile.tac*]

DESCRIPTION

The **melac** command accepts as input Humdrum `**semits` data and outputs a series of values representing the degree of melodic accent associated with each note. Melodic accent values vary between 0 (no accent) and 1 (maximum accent). Input is limited to only a single `**semits` data spine.

The **melac** command implements a model of melodic accent developed by Joseph Thomassen (see REFERENCES). Thomassen's model is sensitive to pitch contour only — distinguishing just three types of melodic motion: ascending, descending, and unison. The model calculates tonal accent values according to a moving 3-pitch window.

It is recommended that output files produced using the **melac** command should be given names with the distinguishing '.mac' extension.

OPTIONS

The **melac** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

The following example illustrates the output of the **melac** command. The `**semits` spine is the input, and the `**melac` spine is the corresponding output. (A `**kern` equivalent to `**semits` has been added to increase the readability.)

```
**kern    **semits    **melac
16ee      16          1
16cc      12          0.5
16b       11          0.355
16cc      12          0.2407
```

```
16g     7       0.1207
16cc    12      0.2407
16b     11      0.1207
16cc    12      0.0957
16ff    17      0.5561
16cc    12      0.085
16b     11      0.355
16cc    12      0.2407
16a     9       0.1207
16cc    12      0.2407
16b     11      0.1207
16cc    12      0.29
*-      *-      *-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

LIMITS

This command is currently able to handle only a single (monophonic) input stream.

REFERENCES

Joseph Thomassen, "Melodic accent: Experiments and a tentative model," *Journal of the Acoustical Society of America,* Vol. 71, No. 6 (1982) pp.1598-1605; see also, Erratum, *Journal of the Acoustical Society of America,* Vol. 73, No. 1 (1983) p.373.

NAME

metpos — assign metric position indicators to sonorities

SYNOPSIS

metpos [**-m** *n*/*n*] [**-t** *n*] [*inputfile.tb* ...] [> *outputfile.met*]

DESCRIPTION

The **metpos** command is used to characterize the metric strength of successive sonorities in a musical passage. Specifically, **metpos** appends a Humdrum `**metpos` spine to a time-base (`*tb`) input stream. The `**metpos` spine contains integer values indicating the position in the metric hierarchy for each data record — given some meter signature.

The highest position in any given metric hierarchy is given the value "1." This value is assigned to the first event at the beginning of each measure. In duple and quadruple meters, the second level in the metric hierarchy occurs in the middle of the measure and is assigned the output value "2." (In triple meters, the second and third beats in the measure will be assigned to the second level in the metric hierarchy.) All other metric positions in the measure (beats, sub-beats, sub-sub-beats, etc.) are assigned successively increasing numerical values according to their placement in the metric hierarchy. Larger `**metpos` values signify sonorities of lesser metric significance.

Input to **metpos** can be any Humdrum file that conforms to the *time-base* format — i.e. where each data record represents an equivalent duration of time. Each input measure will thus contain the same number of data records. For example, in 3/4 meter with an eighth-note time-base, each complete measure should contain 6 data records.

For correct operation, **metpos** must be informed of both the *meter signature* and the *time-base* for the given input passage. This information may be specified either via the command line, or as encoded interpretations in the input stream. The command line method of specification is illustrated below:

```
metpos -m 9/16 -t 32
```

This command establishes a meter of 9/16 and a time-base of a thirty-second duration for the input.

Alternatively, the meter signature and time-base duration can be made known through the presence of interpretation records in the encoded input. In the above case, the meter signature can be made known to **metpos** through the `*M9/16` interpretation, while the time-base duration can be made known to through the `*tb32` interpretation. A time-base and meter signature interpretation must appear in all spines of the input file prior to the occurrence of any data records. Note that once the initial meter is established, **metpos** is able to adapt to encoded changes of meter within a given score or input stream. If meter or time-base information is not available to **metpos** an error message will be issued and execution terminated.

As noted above, metric hierarchies are represented through a series of integer values. The smallest numerical values represent events having the highest metric stress, whereas successively larger values represent positions of progressively weaker metric stress. For example, in the case of 2/4 meter with an 8th duration time-base, the metric hierarchy is: 1, 3, 2, 3. If the time-base is 16th durations, the metric hierarchy is: 1, 4, 3, 4, 2, 4, 3, 4. Metric hierarchies in compound meters are also possible. For example, in the case of 6/8 meter (16th durations) the metric hierarchy is: 1, 4, 3, 4, 3, 4, 2, 4, 3, 4, 3, 4.

Note that **metpos** is unable to deal with irregular meters. (See "LIMITS" below.)

It is recommended that output files produced using the **metpos** command should be given names with the distinguishing '.met' extension.

OPTIONS

The **metpos** command provides the following options:

-h	displays a help screen summarizing the command syntax
-m *n/n*	set the initial meter signature to *n/n*
-t *n*	set the initial time-base duration to *n* (`**recip` value)

Options are specified in the command line.

SAMPLE OUTPUTS

The following extract from Bartók's "Two-Part Study" No. 121 from *Mikrokosmos* demonstrates the effect of the **metpos** command. The two left-most columns show the original input; all three columns show the corresponding output from **metpos**:

```
**kern   **kern   **metpos
*tb8     *tb8     *tb8
=16      =16      =16
*M6/4    *M6/4    *M6/4
8Gn      8b-      1
8A       8ccn     4
8B-      8cc#}    3
8cn      {8f#     4
8c#}     8gn      3
{8F#     8a       4
8G       8b-      2
8A       8ccn     4
```

```
8B-     4b-     3
8cn     .       4
8c#}    8fn}    3
8r      8r      4
=17     =17     =17
*M4/4   *M4/4   *M4/4
8d      2r      1
4.d     .       4
.       .       3
.       .       4
{2d_    8dd     2
.       4.dd    4
.       .       3
.       .       4
=18     =18     =18
8d      {1dd_   1
8A      .       4
8F#     .       3
8E      .       4
8D      .       2
8BB     .       4
8D      .       3
8E}     .       4
=19     =19     =19
*M3/2   *M3/2   *M3/2
{8F#    8dd     1
8A      8ffn    4
8c#     8aa     3
8A      8ff     4
8F#     8dd     2
8A      8ff     4
8F#     8dd     3
8E      8ccn    4
8D      8b-     2
8BBn    8gn     4
8D      8b-     3
8E}     8cc     4
=20     =20     =20
*-      *-      *-
```

Notice that **metpos** adapts to changing meter signatures, and correctly distinguishes between metric accent patterns such as 6/4 (measure 16) and 3/2 (measure 19).

WARNINGS

It is possible to define a time-base that makes little sense with respect to the meter signature. For example, it is possible to set the time-base to a quarter duration (`*tb4`) in a 6/8 meter (`*M6/8`). The resulting metric heirarchy (1,3,3) will produce "hemiola" values.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****kern** (2), ****recip** (2), **timebase** (4)

LIMITS

In a number of meters, **metpos** is limited in the depth of permissible metric positions. These limitations are tabulated below:

compound duple	7 metric levels	96 metric positions/measure
compound triple	6 metric levels	108 metric positions/measure
compound quadruple	7 metric levels	96 metric positions/measure
simple triple	7 metric levels	96 metric positions/measure
simple duple	no limitation	no limitation
simple quadruple	no limitation	no limitation

By way of example, it is possible to process music in 6/8 meter in time-base divisions as small as a 256th-notes, or a passage in 6/2 meter in 64th-note divisions. Smaller time-base divisions are not handled.

The following meter signatures can be handled by **metpos:**

2/2,	3/2,	4/2,	6/2,	9/2,	12/2,
2/4,	3/4,	4/4,	6/4,	9/4,	12/4,
2/8,	3/8,	4/8,	6/8,	9/8,	12/8,
2/16,	3/16,	4/16,	6/16,	9/16,	12/16,
2/32,	3/32,	4/32,	6/32,	9/32,	12/32,
2/64,	3/64,	4/64,	6/64,	9/64,	12/64,

Some meter signatures containing dotted durations can also be handled:

2/2.(=6/4),	3/2.(=9/4),	4/2.(=12/4),
2/4.(=6/8),	3/4.(=9/8),	4/4.(=12/8),
2/8.(=6/16),	3/8.(=9/16),	4/8.(=12/16),

Meter signatures NOT handled by **metpos** include:

5/2,	7/2,	8/2,	10/2,	11/2,	13/2,	14/2,	15/2,	16/2,
5/4,	7/4,	8/4,	10/4,	11/4,	13/4,	14/4,	15/4,	16/4,

5/8,	7/8,	8/8,	10/8,	11/8,	13/8,	14/8,	15/8,	16/8,
5/16,	7/16,	8/16,	10/16,	11/16,	13/16,	14/16,	15/16,	16/16,
etc.								

PROPOSED MODIFICATIONS

Irregular meters such as 5/4 or 7/8 should be acceptable provided the beat subdivisions are made explicit: e.g. 5/4 as (3+2)/4 or 5/4 as (2+3)/4 or 8/8 (3+3+2)/8, etc. Alternatively, no sub-grouping of irregular meters into beats of 2's or 3's need be assumed. Thus, for example, all beats other than the first beat in 5/4 could be assigned identical values in the metric hierarchy.

NAME

midi — convert from `**kern` to Humdrum `**MIDI` format

SYNOPSIS

midi [**-Ccu**] [**-d** *n*] [**-q** *n*] [*inputfile.krn* ...] [> *outputfile.hmd*]

DESCRIPTION

The **midi** command converts Humdrum `**kern` data into Humdrum `**MIDI` data. Humdrum `**MIDI` data may be played by the **perform** command or converted to a standard MIDI file using the **smf** command. The **midi** command will translate all `**kern` spines in the input stream into `**MIDI` data, and will echo any non-kern spines.

It is recommended that output files produced using the **midi** command should be given names with the distinguishing '.hmd' extension ('Humdrum midi').

OPTIONS

The **midi** command provides the following options:

-c	echo all data records as global comments (prior to the data record)
-C	echo all data records as global comments (following the data record)
-d *n*	assigns a note-duration of *n* `**recip` value to all pitches and rests
-h	displays a help screen summarizing the command syntax
-q *n*	set number of MIDI clock ticks per quarter-duration to *n*
-u	suppress the deletion of duplicate (unison) concurrent note-on instructions

Options are specified in the command line.

If the **-c** option ('comment') is invoked, all data records are echoed as global comments in addition to the usual processing. This option is useful in conjunction with the **perform** command; **perform** echoes all global comments while playing MIDI inputs, and so the comment option allows the user to view the original `**kern` score while the music is performed.

The **-d** *n* option allows the user to reassign all the durations of all notes and rests in a given input. The value *n* is specified in reciprocal duration notation (`**recip`), where '4' represents a quarter-duration, '2.' represents a dotted half-duration, '12' represents a quarter-note triplet, '0' represents a breve, etc. (See `**recip` in Section 2 for details.) With the **-d** option all pitches and rests in the input will be assigned the specified duration value — whether or not the notes already have encoded durations. This option is useful for such tasks as proof-listening to passages containing especially short notes, or auditing works — such as Gregorian chant — where durations have not been encoded or are suspect for other reasons. Note that the **-d** option should be used only in the case of monophonic inputs, or multi-part inputs that contain strictly isochronous chords. Polyphonic inputs containing non-isochronous durations will cause a loss of synchronization between the parts. The resulting output is apt to cause serious problems with commands such as **perform** and **smf.**

In contrast to MIDI performance data, canonical musical scores (such as `**kern`) frequently contain unisons — where two voices share the same pitch for a period of time. MIDI produces a 'note-off' instruction at the end of each note; however only a single 'note-off' instruction is required to turn-off a note. Unfortunately, the first note-off will terminate a note — even if the same pitch is sustained in another voice. In normal operation, **midi** suppresses the sending of the first of two MIDI 'note-off' commands when a unison is encountered in the input. This ensures that the pitch will be sustained for the longer of the unison durations. This feature can be disabled by invoking the **-u** option.

The **midi** command assigns a default duration of 72 MIDI clock-ticks per quarter-note. This default value may be reassigned using the **-q** *n* option. The value *n* should be chosen so that all canonical durations present in a given score divide evenly into it. For example, the default value of 72 clock-ticks per quarter-note, means that eighth-notes and sixteenth-notes will have durations of 36 and 18 clock-ticks respectively. Eighth-note triplets and sixteenth-note triplets will have durations of 24 and 12 clock-ticks respectively. Thirty-second notes will have a duration of 9 clock-ticks. But sixty-fourth notes can cause synchronization problems since they do not divide evenly into 72. The user may wish to reassign this value depending on the types of canonical durations present in a given input. Changing the default number of clock-ticks will affect the tempo of performance when using the **perform** command. Hence, it may be necessary to revise the tempo indication of a resulting Humdrum midi-format file.

EXAMPLES

The following examples illustrate how **midi** may be invoked.

```
midi chopin > chopin.hmd
```

converts the `**kern` data from the file `chopin` to `**MIDI` data in the file `chopin.hmd`. The command:

```
midi -c siegfrd.idl > siegfrd.hmd
```

translates the `**kern` data from the file `siegfrd.idl` to `**MIDI` data in the file `siegfrd.hmd` and echoes all `**kern` data as global comments.

SAMPLE OUTPUTS

The operation of **midi** is illustrated in the following inputs and outputs. In the first example, a simple C major scale is encoded in `**kern`.

```
!! midi example #1
**kern
*M2/4
*C:
=1
8c
8r
8d
8e
=2
8f
8g
8a
8b
=3
4cc
4r
====
*-
```

Executing the **midi** command with the default settings results in the `**MIDI` output given below. Notice that a default channel of MIDI channel 1 has been assigned via the tandem interpretation `*Ch1`. Note also that each `**kern` note has resulted in two MIDI instructions: "key-on" and "key-off". Since the end of one note coincides with the start of the subsequent note, key-on and key-off data are output as Humdrum double-stops (two tokens separated by a single space character).

```
!! midi example #1
**MIDI
*Ch1
*M2/4
*C:
=1
72/60/64
36/-60/64
36/62/64
36/-62/64 36/64/64
=2
36/-64/64 36/65/64
36/-65/64 36/67/64
36/-67/64 36/69/64
36/-69/64 36/71/64
=3
36/-71/64 36/72/64
72/-72/64
====
.
*-
```

`**MIDI` data consist of three numbers separated by slashes (/). The first integer represents the number of clock ticks that must elapse from the previous instruction before the current event is initiated. In the above case, a default value of 72 clock ticks per quarter-note has resulted in 36 clock ticks for each eighth note. The second integer represents the MIDI key number, where middle C is represented as key #60 (negative numbers denote key-off instructions). The third integer represents the MIDI key velocity. The default key velocity is 64 units.

The second example illustrates the handling of input containing multiple-stops.

```
!! midi example #2
**harm   **kern
*        *Ch3
=1       =1
I        4c 4e 4g
IV       4c 4f 4a
V        4d 4g 4b
I        4e 4g 4cc
==       ==
*-       *-
```

The output below is generated by invoking the following command:

```
midi -c input > output
```

```
!! midi example #2
**harm     **MIDI
*          *Ch1
!!**harm   **kern
*          *Ch3
!!*        *Ch3
=1         =1
!!=1       =1
I          0/60/64 0/64/64 0/67/64
!!I        4c 4e 4g
IV         96/-60/64 96/-64/64 96/-67/64 96/60/64 96/65/64 96/69/64
!!IV       4c 4f 4a
V          96/-60/64 96/-65/64 96/-69/64 96/62/64 96/67/64 96/71/64
!!V        4d 4g 4b
I          96/-62/64 96/-67/64 96/-71/64 96/64/64 96/67/64 96/72/64
!!I        4e 4g 4cc
==         ==
!!==       ==
.          96/-64/64 96/-67/64 96/-72/64
*-         *-
!!*-       *-
```

Notice that non-kern data (`**harm`) is echoed in the output. Also, notice that each input record has been reproduced as a global comment (preceded by !!). This feature is useful in conjunction with the **perform** command.

DIAGNOSTICS

The **midi** command echoes tempo indications for the benefit of the **perform** command. If a tempo range is specified (e.g. MM92-98), **midi** calculates the average range and echoes that (MM95). Tempo terms (such as "largo") are not handled by **midi.**

PORTABILITY

DOS 2.0 and up.

SEE ALSO

humdrum (4), ****kern** (2), **kern** (4), ****MIDI** (2), **perform** (4), **proof** (4), **smf** (4), **tacet** (4)

PROPOSED MODIFICATIONS

Channel assigment tandem interpretations (e.g. `*Ch6`) should be recognized in the input stream and cause the default channel 1 output to be suppressed.

NAME

midreset — reset MIDI controller card

SYNOPSIS

midreset

DESCRIPTION

The **midreset** command resets the MIDI controller card. The **midreset** command is useful if a MIDI application has terminated abnormally — leaving the MIDI card unaccessable to further use.

OPTIONS

The **midreset** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

PORTABILITY

DOS 2.0 and up.

SEE ALSO

perform (4), **smf** (4), **tacet** (4)

REFERENCES

Use of the Music Quest Inc. MIDI library functions is gratefully acknowledged.

NAME

mint — determine melodic intervals between successive pitches

SYNOPSIS

mint [**-acd**] [**-b** *regexp*] [**-s** *regexp*] [*inputfile*] [> *outputfile.mnt*]

DESCRIPTION

The **mint** command determines the distance (interval) between successive pitches. Output pitch intervals are expressed as a diatonic interval size plus interval quality; a leading plus or minus sign indicates whether the interval is ascending or descending. By way of illustration, **mint** will change a sequence of `**pitch` data tokens — such as *C4, A4, E4* — to the interval sequence *+M6, –P4.* Each pitch-related input spine is transformed to a corresponding `**mint` output spine.

The **mint** command determines melodic intervals only for pitch tokens *within* individual spines. Pitch intervals *across* spines are not determined by **mint.**

No interval is calculated for the first pitch token; initial pitches are simply echoed in the output — appearing in square brackets. These initial pitches are referred to as *offset* values, since they indicate the starting value from which subsequent intervals are calculated. Offset values can prove useful in attempting to reconstruct the input, but the user may wish to eliminate offset values in subsequent processing (see below).

The **mint** command is able to handle multiple-stops. Data-flow interruptions such as the occurrence of barlines can be handled using the **-s** option. By defining *regular expression* patterns, the user may select which types of data tokens should be ignored by **mint.** (See EXAMPLES below.)

Note that the output spine generated by **mint** preserves the same record-type structure as the input, and so may readily be pasted with the input file using the Humdrum **assemble** command.

The **mint** command is able to accept any of the pitch-related representations listed below. For descriptions of the various input representations refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **mint** command should be given names with the distinguishing '.mnt' extension.

`**kern`	core pitch/duration representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**solfg`	French solfège system (fixed 'doh')
`**Tonh`	German pitch system

Input representations processed by **mint**.

OPTIONS

The **mint** command provides the following options:

-a	output absolute pitch interval without distinguishing ascending/descending
-b *regexp*	break; do not calculate difference for tokens matching *regexp*; restart interval calculations with next pitch token
-c	output compound intervals as non-compound intervals
-d	output diatonic interval size only, without the interval quality
-h	displays a help screen summarizing the command syntax
-s *regexp*	skip; completely ignore tokens matching *regexp*; (echo in output only)

Options are specified in the command line.

By default, **mint** distinguishes ascending and descending intervals by prepending a plus or minus sign. If the **-a** option is invoked, then only unsigned (absolute) intervals are output.

The "skip" function takes precedence over the "break" function, so input strings matching both the skip (-s) and break (-b) regular expressions cause a skip rather than a break.

EXAMPLES

The various aspects of the **mint** command are best illustrated using a set of examples. Consider the following input:

```
**kern
=1
8c
8g
4.b-
.
8e
=2
4f
8r
8C
4FF
=3
*-
```

Using the default invocation, the **mint** command transforms the above input as follows:

```
**mint
=1
[c]
+P5
+m3
.
-d5
=2
+m2
r
-P11
-P5
=3
*-
```

The leading or "offset" pitch 'c' has been echoed in square brackets in the third record. This represents the initial pitch from which subsequent pitch distances are measured. This offset value reflects the type of input given to **mint,** thus if the input format is `**kern` the offset pitch will be recorded in the same representation. Note that for absolute pitch units: 'c' (`**kern`) equals 'C4' (`**pitch`) equals 'do4' (`**solfg`) equals 'C4' (`**Tonh`).

The subsequent output value (+P5) indicates a melodic interval of an ascending perfect fifth. This is followed by an ascending minor third (+m3) followed by a descending diminished fifth (-d5).

Notice that the null-token in the sixth record has been echoed. Null-tokens have no effect on interval calculations and are treated as though they are non-existent. In addition, **mint** correctly echoes (and ignores) both rests and kern-like barlines. Note that pitch intervals spanning a rest continues to be calculated and that intervals greater than an octave remain as "compound intervals."

Depending on the application, users may wish to suppress the calculation of intervals across rests. In this case, the **mint** command can be invoked using the "break" (**-b**) option. Consider the command: `mint -b 'r' input > output.mnt`. Each time **mint** encounters a data token that matches the letter 'r' it echoes the input token and begins looking for a new offset value. Applied to the earlier input file, this command produces the following output:

```
**mint
=1
[c]
+P5
+m3
.
-d5
=2
+m2
8r
[C]
-P5
=3
*-
```

Notice that the descending perfect eleventh spanning the rest has been eliminated, and a new pitch offset value 'C' has been echoed in the corresponding output.

The **-s** (skip) option can be used to allow the user to selectively identify records that should not be involved in processing. For example, the command

```
mint -s '^[^4]+$' input > output.mnt
```

will cause any data token not matching the number 4 to be skipped during processing. Given the sample input, intervals will be calculated only between quarter-notes and dotted quarter-notes:

```
**mint
=1
8c
8g
[b-]
.
8e
=2
-P4
8r
8C
-P15
=3
*-
```

Using the skip option, the user may calculate melodic intervals between pitches in strong metric positions, or pitches that have been marked as structural tones.

The **mint** command is also able to process numerical data tokens containing multiple-stops. Consider the following following sample input:

```
**pitch
C4
B3 D4
A3 C4 F4
C4
C4
*-
```

Notice the presence of the double- and triple-stops in the third and fourth records. Using the default invocation, the **mint** command processes this input as follows:

```
**mint
[C4]
-m2 +M2
-M2 (+m2) (-M2) +m3
+m3 P1 -P4
P1
*-
```

The leading or offset value [C4] has been echoed in the second record. (The user might wish to eliminate such offset values via the **humsed** command; see below.) The third record in both the input and output contain double-stops. In the output, the first value of the double-stop (-m2) represents the pitch interval between C4 and B3. The second value in the double-stop (M2) represents the difference between C4 and D4. In short, **mint** traces both possible difference "paths."

In processing successive multiple-stops **mint** does not calculate all of the possible permutations. For example, in the case of two consecutive triple-stops, **mint** will calculate three intervals corresponding to the first notes in both triple-stops, the second notes, and the third notes.

Where the number of multiple-stops changes, **mint** operates under some special conventions. Consider, for example, the case of a double-stop followed by a triple-stop: the pitches P+Q followed by X+Y+Z. All of the possible (interval) differences might be enumerated as follows: X-P, Y-P, Z-P, X-Q, Y-Q and Z-Q. **Mint** first calculates the "outer" interval distances (X–P and Z–Q). It then calculates a permuted set of "inner" intervals (Y–P and Y–Q). The remaining intervals are considered unlikely or implausible and are not calculated by **mint.**

In the above example, moving from the double-stop to the triple stop between records three and four generates two "outer" interval distances (B3 to A3 → -M2; D4 to F4 → +m3), as well as the permuted "inner" intervals (B3 to C4 → +m2; D4 to C4 → -M2). Both the resulting inner intervals are printed in parentheses. A similar process occurs when moving from records four to five. Three intervals may be traced from the 3 initial pitches to the subsequent single pitch.

Depending on the goal, the presence of the parentheses makes it easy for the user to eliminate the inner intervals using the Humdrum stream-editor **humsed.** For example, the command

```
humsed 's/(([^)]*) //g' input > output
```

can be used to eliminate inner intervals. Alternatively, the command

```
humsed 's/[()]//g' input > output
```

can be used to eliminate the parentheses surrounding the inner intervals. Offset values can be transformed to null-tokens using the command

```
humsed 's/\[[^]]*\]/./g' input > output
```

records containing offset values can be eliminated using the command

```
humsed '/\[.*\]/d' input > output
```

One final example illustrates how several spines can be processed concurrently by **mint.**

```
!! J.S. Bach, keyboard Sinfonia No. 13
**Tonh          **pitch         **kern
*M3/8           *M3/8           *M3/8
*a:             *a:             *a:
=7              =7              =7
A3              G4              16ee
.               .               16ff#
H3              F#4             8.dd#
H2              .               .
.               .               16ee
=8              =8              =8
E3              E4              4ee
.               F#4             .
.               G4              .
.               A4              .
r               B4              [8gg
.               C#5             .
=9              =9              =9
r               [D5             16gg]
.               .               16bb-
.               .               16aa
.               .               16gg
.               .               16ff
.               .               16ee
=10             =10             =10
r               D5]             [4.ff
*-              *-              *-
```

The following command invokes the **-a** and **-d** options. Indications of the direction of interval movement (ascending/descending) have been removed, and the diatonic interval sizes are output without the associated interval qualities (major/minor/perfect/diminished, augmented).

```
mint -a -d input | humsed 's/\[[^\t]*\]/./g' > output
```

Notice in the corresponding output that the initial offset pitches have been changed to a null tokens (due to the **humsed** command).

```
!! J.S. Bach, keyboard Sinfonia No. 13
**mint          **mint          **mint
*M3/8           *M3/8           *M3/8
*a:             *a:             *a:
=7              =7              =7
.               .               .
.               .               2
2               2               3
8               .               .
.               .               2
=8              =8              =8
4               2               1
.               2               .
.               2               .
.               2               .
r               2               3
.               2               .
=9              =9              =9
r               2               1
.               .               3
.               .               2
.               .               2
.               .               2
.               .               2
=10             =10             =10
r               1               2
*-              *-              *-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****hint** (2), **hint** (4), **humsed** (4), **kern** (2), ****mint** (2), **recode** (4), **regexp** (6), ****semits** (2), **semits** (4), **solfg** (2), ****Tonh** (2), **xdelta** (4), **ydelta** (4)

NAME

nf — determine normal form for successive vertical sonorities

SYNOPSIS

nf [*inputfile* ...] [> *outputfile.nf*]

DESCRIPTION

The **nf** command is used to determine the *normal form* for any of five set-theory related inputs: pitch (`**semits`), pitch-class (`**pc`), prime form (`**pf`), interval-vector (`**iv`), or Fortean set name (`**pcset`).

"Normal form" is a standardized way of representing the pitch material for any arbitrary set of pitches. Normal form provides the most intervallically compact spelling of the pitch-classes evident in a given sonority. It is analogous to rearranging notes in a chord so that the spelling is in root position, close position, with duplicate pitch-classes eliminated. By way of example, a D major chord (in any inversion, with any spelling) will have the normal form: 2,6,9 — namely, the pitch-classes D, F#, A (as opposed to 6,9,2 or 2,9,6). See REFERENCES below.

When provided with `**semits` or `**pc` inputs, **nf** treats each input record as a set of pitches. Unisons and other pitch-class duplications have no effect on the output. Rests within a set of pitches are ignored; where an input record consists solely of one or more rests, a null-token is output.

The **nf** command is able to translate any of the representations listed below. For descriptions of the various input representations (including `**nf`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **nf** command should be given names with the distinguishing '.nf' extension.

`**iv`	interval vector representation
`**nf`	normal form representation
`**pc`	pitch-class representation
`**pcset`	Fortean pitch-class set name
`**pf`	prime form representation
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)

Input representations processed by **nf**.

OPTIONS

The **nf** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

The following command outputs the normal form for the sets formed by successive sonorities in the input file `opus24`. The input may be pitches, pitch-classes, Fortean set names, etc.

```
nf opus24 > opus24.nf
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

context (4), ****iv** (2), **iv** (4), ****nf** (2), ****pc** (2), **pc** (4), ****pcset** (2), **pcset** (4), ****pf** (2), **pf** (4), ****semits** (2), **semits** (4)

NOTE

The **nf** command is a shell script that invokes `pcset -n`.

REFERENCES

Allen Forte, *The Structure of Atonal Music.* New Haven: Yale University Press, 1973.

John Rahn, *Basic Atonal Theory.* New York: Longman Inc., 1980.

Straus, J. *Introduction to Post-Tonal Theory.* Englewood Cliffs, N.J.: Prentice Hall, 1990.

NAME

num — number selected records according to user-defined criteria

SYNOPSIS

num [**-efT**] [**-a** ***interp*] [**-i** *n*] [**-n** *regexp*] [**-o** *n*] [**-O** *n*] [**-p** *regexp*] [**-P** *regexp*] [**-r** *regexp*] [**-R** *regexp*] [**-s** *regexp*] [**-S** *regexp*] [**-x** *regexp*] [**-z** *regexp*] [**-Z** *regexp*] [*inputfile* ...]

DESCRIPTION

The **num** command produces sequential numerical output according to user-defined numbering criteria. In its default operation, **num** simply inserts numbers at the beginning of each data record — beginning with the value 1, and increasing by 1 for successive data records. However, **num** provides various options that allow the user to specify more precisely the kinds of data and conditions under which numbering occurs.

Numbers may be inserted prior to data tokens, appended following data tokens, or inserted in the middle of data tokens. Numbers may be output only for the first input spine, or for all input spines. Alternatively, numbers may be output in a separate output spine specified by the user. Numbers may be assigned only to those data records matching a given regular expression, or may be assigned to all records *other* than those matching a regular expression. Numerical counts may begin at any real or integer value and may be incremented or decremented by any real or integer value. Numbering may restart at some defined value whenever a certain regular expression is matched in the input. Numbers may be output only when certain conditions are met; for example, although counting may continue, outputting of numbers may be suspended or resumed when the input data match user-specified regular expressions.

The **num** command may be used to number measures, phrases, chords, notes, rests, or other musically-pertinent features. (See EXAMPLES below.)

OPTIONS

The **num** command provides the following options:

-a ***interp*	append a new spine (***interp*) containing numbered data
-e	place numbers at end of data tokens (rather than at the beginning)
-f	number all spines (all fields) in the input
-h	displays a help screen summarizing the command syntax
-i *n*	set increment value to *n*
-n *regexp*	number only those records matching *regexp*
-o *n*	set initial offset value to *n*
-O *n*	set offset value to *n* after a reset
-p *regexp*	place output number immediately following the first occurrence of *regexp* on the line
-P *regexp*	place output number immediately following the first occurrence of *regexp* on the line
-r *regexp*	resume numbering records when *regexp* is matched
-R *regexp*	resume numbering records **after** *regexp* is matched
-s *regexp*	suspend numbering records when *regexp* is matched
-S *regexp*	suspend numbering records **after** *regexp* is matched
-T	reset counter when all spines have exclusive interpretations
-x *regexp*	exclude numbering those records matching *regexp*
-z *regexp*	reset counter when record matches *regexp*
-Z *regexp*	reset counter **after** record matches *regexp*

Options are specified in the command line.

Normally, the effect of **num** is to add numbers to data tokens already in the input. With the **-a** option, **num** creates a new spine which is appended to the right of the input stream. The numerical outputs of **num** become data records in this new output spine. The user can specify the name of the output interpretation via the command line parameter ***interp*. The **-a** cannot be used with the **-p** or **-P** options.

The **-e** option causes numbers generated by **num** to be append to the end of each appropriate data token rather than the (default) beginning of each data token. The **-e** and **-a** options are mutually exclusive. In addition, the **-e** cannot be used with the **-p** or **-P** options.

The **-f** option causes all spines (all fields) in the input to be numbered rather than the (default) first (left-most) spine. The **-f** option is mutually exclusive with the **-a** option. In addition, the **-f** cannot be used with the **-p** or **-P** options.

The **-i** option allows the user to set the increment value for successive numbers. The default value is 1 — meaning that successive numerical outputs are 1 greater than the previous value. Negative increment values are also permissible. For example, the user might define an initial value beginning at 100, and decrement by 5 with each successive value.

The **-n** option causes **num** to output numerical values, only when the current data record matches a specified regular expression.

The **-o** option is used to define an initial (offset) value from which subsequent numbers are calculated. If no offset is defined, the default value is 1.

The **-O** option defines an offset value to which the counter will be returned each time a *reset* action occurs. The **-O** option should be used in conjunction with one of either the **-T**, **-z** or **-Z** options.

The **-p** and **-P** options allow the user to place any output numerical value in a particular (horizontal) place in the output line. In the case of **-p** the output number is positioned immediately following the first (left-most) string matching the specified regular expression. With the **-P** option, the output number is positioned immediately prior to the first string matching the specified regular expression. The **-p** and **-P** options cannot be used with either the **-a**, **-e** or **-f** options.

The **-r** option defines a condition under which the outputting of numbers will resume. Specifically, the user defines a regular expression with the **-r** option that, when matched, causes the immediate resumption of printing.

The **-R** option is similar to the **-r** option, with the exception that outputting of numbers is resumed *after* any record matching the specified regular expression.

The **-s** option causes the outputting of numbers to be suspended when an input record matches a specified regular expression. Although the numerical values are not outputted, the numerical values continue to be incremented in accordance with the defined counting conditions.

The **-S** option is similar to the **-s** option, with the exception that the outputting of numbers is suspended *after* any record matching the specified regular expression.

The **-T** option causes the counter to be reset (to the value specified by **-O**) whenever exclusive interpretations are encountered in all of the input spines. If no initial offset has been specified via the **-O** option, then the counter is reset to the value 1.

The **-x** option causes records matching a given regular expression to be excluded from the counting; no output is generated for such records. Note that when used in conjunction with the **-n** option, both the *match* and *don't match* criteria must be fulfilled in order for the current record to participate in the counting.

The **-z** option causes the counter to be reset (to the value specified by **-O**) whenever a data record matches a specified regular expression. If no initial offset has been specified via the **-O** option, then the counter is reset to the value 1.

The **-Z** option is similar to the **-z** option, with the exception that the counter is reset *after* any record matching the specified regular expression.

EXAMPLES

The following examples illustrate how **num** may be used. Consider the following input (left spine) and corresponding **num** output (right spine).

```
**kern   **plength
=23      .
{8a      .
.        .
8cc      .
}8ee     3
{8g#     .
=24      .
8dd      .
8ee      .
}8ff     4
8r       .
.        .
=25      .
{8gn     .
8cc      .
}8ee     3
{8f#     .
=26      .
8cc      .
8dd      .
}8ee-    4
*-       *-
```

The `**plength` output indicates the number of notes in each phrase for the corresponding `**kern` spine. The output was generated using the following command:

```
num -a '**plength' -z '{' -x '[.r=]' -s '{' -r '}' -S '}'
```

The **-x** option excludes `**kern` rests, barlines, and null tokens from the counting. The **-z** option causes the counter to be reset to 1 whenever a begin-phrase signifier ('{') is encountered. The **-s** option causes suspension of output numbers to occur at the beginning of each phrase, and the **-r** option causes output numbers to be resumed at the end of each phrase (hence, only the phrase-end signifiers are given numbered output). The **-S** option ensures that numbers are not printed for notes outside of phrases; that is, it suspends outputting numbers following the end of a phrase.† The **-a** option causes the numbers to be output as a separate spine labelled `**plength`.

The command

```
num -a '**ordo' koto
```

outputs a new spine labelled `**ordo` containing successive integers beginning at 1 for each

† Note that this command will still fail to suppress the numbering of notes occuring prior to the first phrase.

data record in the input.

```
num -n '^=' -x '==' -p '=' -o 108 sarod
```

numbers all "common system" barlines in the file `sarod`, beginning with measure 108. Double barlines are not numbered (due to the **-x** option) and numbers are positioned directly following the equals sign (due to the **-p**) option). The **-p** option ensures that the number precedes pause markings and other possible barline signifiers. Note that if measure numbers already exist for a file, the measures can be renumbered by first removing the current measure numbers using **humsed**.

The command

```
num -a '**phrase#' -n '{' -T rebec
```

outputs a spine containing numbers that number the beginning of each `**kern` phrase for the file `rebec`; if any exclusive interpretation is encountered, the phrase numbering restarted at 1.

The command

```
num -x '^='
```

numbers all data records other than common system barlines.

```
num -x '^=' -Z '='
```

numbers all data records within each common system measure — starting at the value 1 with each new measure.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

nl (UNIX), ****ordo** (2), **regexp** (4), **regexp** (6), **rend** (4)

NOTES

The **-O** option should be used in conjunction with one of either the **-T**, **-z** or **-Z** options.

NAME

patt — locate and output user-defined patterns in a Humdrum input

SYNOPSIS

patt [**-m**] **-f** *templatefile* [**-s** *regexp*] [**-t** *output_tag*] [*inputfile* ...]

DESCRIPTION

The **patt** command is used to locate occurrences of a user-defined pattern in some Humdrum input. The patterns sought may span innumerable data records. Occurrences of the pattern may be identified by line number, echoed intact in the output stream, or tagged by a user-defined marker in a `**patt` output spine.

The pattern sought must be defined as a separate "template" file. The template file is identified using the **-f** command option.

Pattern templates consist of one or more records. Each record specifies a regular expression pattern. The input is scanned from beginning to end, in order to find passages that match the defined template. In order for a match to take place, successive records in the input stream must match the regular expressions given in each of the corresponding records of the template. For example, if the template is 9 lines in length, then the input stream must contain 9 successive lines of matching data in order for a pattern match to be successful.

The **patt** command implements a full UNIX regular expression syntax. Each line in the template file represents an independent regular expression. For example, the template:

```
1
2
3
```

will match inputs such as the following:

```
1
112
43.9
```

or even

```
0x.=%&1*
Figure 32
abc(...32...)
```

A more circumspect regular expression template might look like this:

```
^1$
^2$
^3$
```

(The caret (^) and dollar sign ($) are regular expression *anchors* that indicate the beginning of the record and end of the record respectively.) Refer to *regexp (6)* for further information regarding regular expressions. Note that the related **pattern** command implements a more powerful regular expression syntax for defining multi-record patterns.

OPTIONS

The **patt** command supports the following options:

-c	makes pattern-matching sensitive to comments
-e	echoes matched patterns in the output
-f *templatefile*	use pattern specified in *templatefile*
-h	displays a help screen summarizing the command syntax
-m	invokes collapsed multiple-record matching mode
-s *regexp*	skip (ignore) data records containing the defined regular expression
-t *output_tag*	generate `**patt` output spine; tag each occurrence of the pattern with the string *output_tag*

Options are specified in the command line.

By default, the **patt** command is insensitive to the presence or absence of Humdrum comments. Pattern searches may be made sensitive to occurrences of comments by specifying the **-c** option.

In the default operation, **patt** outputs a Humdrum global comment for each pattern matched in the input. Each comment identifies the line number in the input where the found pattern begins.

With the **-e** option, each instance of the found pattern is echoed in the output. Each output pattern is preceded by the appropriate exclusive interpretation(s) and followed by appropriate spine-path terminator(s).

Certain types of data records may be ignored in the pattern-search by invoking the **-s** (skip) option. This option must be accompanied by a user-define regular expression. All input data records matching the regular expression are ignored. This option is useful, for example, in skipping null data tokens, barlines, marked embellishment tones, or other types of data.

The **-m** option invokes a multiple record matching mode. In this mode, **patt** attempts to match as many successive regular expressions in the template file as possible for a given input record, before continuing with the next input and template records. In this way, a template file consisting of several records, may possibly match a single input record. (See EXAMPLES below.)

The **-t** option causes **patt** to output a single spine of `**patt` output. The user specifies an *output tag* (character string) on the command line. Each instance of the found pattern causes the tag to be output in the `**patt` spine at the position corresponding to the onset of each found pattern. (See EXAMPLES below.) Note that the **-t** and **-e** options are mutually exclusive.

Whatever options are invoked, **patt** always produces output that conforms to the Humdrum syntax.

EXAMPLES

The following examples illustrate the operation of the **patt** command. Consider the following target file and pattern template file:

template file:

```
1
2
3
```

target file:

```
**foo
1
2
=1
1 3
2 1
3 2
3
=2
1
2 3
4
2 3 1
*-
```

A simple search for the pattern template might use the command:

```
patt -f template target
```

Pattern matches are announced by outputting Humdrum global comments. Given the above command, the following output would result:

```
!! Pattern found at line 5 of file input
!! Pattern found at line 6 of file input
```

In the first instance, notice that **patt** is able to identify overlapping patterns. Two instances of the 1-2-3 pattern are identified beginning on the fifth and sixth lines respectively.

Note however, that the first instance of the pattern (beginning at line 2) was not identified due to the interruption of the common system barline in the fourth line. The barlines can be ignored by invoking the **-s** option, followed by a regular expression that uniquely identifies the records to be skipped — in this case the equals sign. The command:

```
patt -s = -f template target
```

would produce the following output:

```
!! Pattern found at line 2 of file input
!! Pattern found at line 5 of file input
!! Pattern found at line 6 of file input
```

Actual instances of the pattern can be output by invoking the **-e** (echo) option:

```
patt -e -s = -f template target
```

The following output would result:

```
!! Pattern found at line 2 of file input
**foo
1
2
=1
1 3
*-
!! Pattern found at line 5 of file input
**foo
1 3
2 1
3 2
*-
!! Pattern found at line 6 of file input
**foo
2 1
3 2
3
*-
```

Notice that each pattern found is output as a self-contained Humdrum output with initial exclusive interpretations and concluding spine-path terminators. If a single continuous output is desired, the **rid -u** command may be used to eliminate the duplicate interpretations.

Instead of outputting the individual patterns, the **-t** option may be used to output a spine that marks each instance of the found pattern. In the following command, the beginning of each occurrence of the pattern is labelled in the `**patt` spine by the tag "one-two-three."

```
patt -t one-two-three -s = -f template target
```

The follow output would result:

```
**foo    **patt
1        one-two-three
2        .
=1       .
1 3      one-two-three
2 1      one-two-three
3 2      .
3        .
=2       .
1        .
2 3      .
4        .
2 3 1    .
*-       *-
```

For some tasks (such as the identification of tone-rows in 12-tone music), nominally "successive" elements of the pattern may be collapsed within a single sonority or record. The **-m** option invokes a *multiple record matching* mode. By way of example, the following command:

```
patt -m -t theme123 -s = -f template target
```

will produce the following output:

```
**foo    **patt
1        theme123
2        .
=1       .
1 3      theme123
2 1      theme123
3 2      .
3        .
=2       .
1        theme123
2 3      .
4        .
2 3 1    theme123
*-       *-
```

Note that in the above examples, the extensive capabilities for defining complex regular expressions have not been used. Refer to **regexp (6)** for further pertinent information.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

grep (UNIX), **egrep** (UNIX), **pattern** (4), **regexp** (4), **regexp** (6), **simil** (4)

WARNINGS

If a comment is present in the template pattern, failing to specify the **-c** option will make pattern matching a logically impossibility.

NOTE

The **patt** command differs from the related **pattern** command in the following ways: (1) **patt** always produces output conforming to the Humdrum syntax whereas **pattern** never produces Humdrum output. (2) **pattern** allows multi-record 'wild cards' in the template file, and so permits the creation of more sophisticated regular expressions. (3) The **pattern** command does not directly provide an "echo" option.

NAME

pattern — exhaustively locate and count user-defined patterns in a Humdrum input

SYNOPSIS

pattern [**-ciy**] **-f** *templatefile* [**-s** *regexp*] [*inputfile* ...]

DESCRIPTION

The **pattern** command is used to locate all occurrences of a user-defined pattern in some Humdrum input. The patterns sought may span innumerable data records. Occurrences of the pattern are identified in the output by line number.

The pattern sought must be defined as a separate "template" file. The template file is identified using the **-f** command option.

Pattern templates consist of one or more records. Each record specifies a standard UNIX regular expression — followed by an optional "record-count metacharacter." The input is scanned from beginning to end, in order to find passages that match the defined template. In the simplest case, a match is deemed to take place when successive records in the input stream match the regular expressions given in each of the corresponding records of the template. However, the number of records in the matching input need not be the same as the number of records in the template.

Consider, first, a simple example where the template consists of the numbers 1, 2, 3 — each on a separate line:

```
1
2
3
```

This template will match inputs such as the following:

```
1
112
43.9
```

or even

```
0x.=%&1*
Figure 32
abc(...32...)
```

A more circumspect regular expression template might look like this:

```
^1$
^2$
^3$
```

(The caret (^) and dollar sign ($) are regular expression *anchors* that indicate the beginning of the record and end of the record respectively.)

Standard regular expression syntax provides three "counting" metacharacters that can be used to specify the number of occurrences of a given pattern on a single line. The counting metacharacters are + * and ?. If *p* is a regular expression pattern, then (*p*)+ will match one or more consecutive instances of *p*. Similarly, (*p*)* will match zero or more consecutive instances of *p*, whereas (*p*)? will match zero or one instance of *p*. The use of these metacharacters is illustrated below:

X+	matches X, XX, XXX, etc.
X*	matches X, XX, XXX, etc. as well as the null string
X?	matches X or the null string
XX	matches XX
(XX)+	matches XX, XXXX, XXXXXX, etc.

Regular expression counting metacharacters

These metacharacters can be used in conjunction with other regular expression operators and anchors to specify complex patterns. See **regexp (6)** for further details.

In the **pattern** command, the regular expression counting metacharacters may *also* be used to specify the *number of successive records* that match the regular expression. We refer to this use as "record-count metacharacters." Record-count metacharacters are specified by following the regular expression with a tab — followed by either +, *, or ?. For example, consider the following **pattern** template:

```
X	+
Y	*
Z	?
```

The intervening tab characters are important here. They indicate that the metacharacters refer to the number of records rather than to the number of patterns in a given record. The first template record (X<*tab*>+) will match one or more lines containing the letter X. The second template record (Y<*tab*>*) will match zero or more lines containing the letter Y. The third template record (Z<*tab*>?) will match zero or one line containing the letter Z. The above template would match an input such as the following: three successive lines containing the letter X, followed by eight successive lines containing the letter Y, followed by a single line containing the letter Z. Similarly, the above template would also match a one-line input containing the letter X.

Note that the strings <*tab*>+ <*tab*>*, and <*tab*>?, are recognized by **pattern** as record-count metacharacters *only* if they appear at the end of a regular expression.

The **pattern** command will identify *all* possible matching patterns beginning at each point in the input. Consider, by way of example, the following template file (named `template`):

```
1   +
2   +
3   +
4   +
5   +
```

The following Humdrum input file is named `example1`:

```
**num   **num
1       1
2       2
3       2
4       3
5       4
5       5
6       6
*-      *-
```

Given the command:

```
pattern -f template example1
```

the **pattern** command will produce the following output:

```
4 patterns found from line 2 to line 7 of file example1
1 pattern found from line 2 to line 6 of file example1
```

The patterns are: 1-2-3-4-5, 1-2-2-3-4-5, 1-2-3-3-4-5, 1-2-3-4-4-5 and 1-2-3-4-5-5. Note that the entire input line is used for matching purposes. It doesn't matter, for example, whether the number "2" is matched in the left spine or the right spine — only that the number "2" is present on a given line. This feature is useful for identifying *Klangfarbenmelodie* and other "threaded" or "diagonal" patterns that can be traced between spines. If the user wishes to avoid such diagonal patterns, individual spines should be extracted separately before invoking the **pattern** command.

OPTIONS

The **pattern** command supports the following options:

-c	makes pattern-matching sensitive to comments
-h	displays a help screen summarizing the command syntax
-i	makes pattern-matching sensitive to interpretations
-s *regexp*	skip (ignore) data records containing the defined regular expression
-y	outputs appropriate 'yank' commands in place of regular output

Options are specified in the command line.

By default, the **pattern** command is insensitive to the presence or absence of Humdrum comments and interpretations. Pattern searches may be made sensitive to occurrences of comments (defined in the template) by specifying the **-c** option. Similarly, pattern searches may be made sensitive to occurrences of interpretations by specifying the **-i** option.

Certain types of data records may be ignored in the pattern-search by invoking the **-s** (skip) option. This option must be accompanied by a user-define regular expression. All input data records matching the regular expression are ignored. This option is useful, for example, in skipping null data tokens, barlines, marked embellishment tones, or other types of data.

The **pattern** command does not directly implement an "echo" option — such as provided by the related **patt** command. With the **-y** option, however, **pattern** will produce an output suitable for use with the Humdrum **yank** command. This permits the user to extract the appropriate matching passages from the input.

EXAMPLES

For additional examples pertinent to the **pattern** command, refer to the EXAMPLES section in the documentation for the related **patt** command.

Note that in the above example, the extensive capabilities for defining complex regular expressions have not been used. Refer to **regexp (6)** for further pertinent information.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

grep (UNIX), **egrep** (UNIX), **patt** (4), **regexp** (4), **regexp** (6), **simil** (4)

WARNINGS

If a comment is present in the template pattern, failing to specify the **-c** option will make pattern matching a logically impossibility.

NOTE

The **pattern** command differs from the related **patt** command in the following ways: (1) **patt** always produces output conforming to the Humdrum syntax whereas **pattern** never produces Humdrum output. (2) **patt** does not support multi-record 'wild cards' in the template file, and so limits the sophistication of the regular expressions. (3) The **patt** command provides "echo" and "tag" options.

NAME

pc — translate semitone representation or pc to pitch-class

SYNOPSIS

pc [**-atx**] [*inputfile* ...] [> *outputfile.pc*]

DESCRIPTION

The **pc** command transforms various pitch-related inputs to corresponding numerical pitch-class equivalents: C=0, C-sharp/D-flat=1, D=2, . . . B=11. It outputs one or more Humdrum `**pc` spines containing pitch-class values corresponding to each of the input pitch tokens. For example, the `**semits` token "–13" is transformed to 11 (pc).

The **pc** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the input and output representations refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **pc** command should be given names with the distinguishing '.pc' extension.

`**cents`	hundredths of a semitone with respect to middle C=0 (e.g. 1200 equals C5)
`**freq`	fundamental frequency (in hertz)
`**kern`	core pitch/duration representation
`**pc`	pitch-class representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **pc**.

Note that the **pc** command is also able to reprocess pitch-class (`**pc`) inputs. This allows pitch-class representations to be translated from numeric-only (0,1,2,3 ... 9,10,11) to mixed alphanumeric (0,1,2,3 ... 9,A,B) or vice versa. (See documentation for `**pc (2)`.)

OPTIONS

The **pc** command provides the following options:

-a output alphanumeric representation (where A=10, B=11)
-h displays a help screen summarizing the command syntax
-t suppresses printing of all but the first note of a group of tied `**kern` notes
-x suppresses printing of non-pitch-class data

Options are specified in the command line.

The **-a** option invokes an alternative (alphanumeric) form of the `**pc` output where the upper-case letters 'A' and 'B' are substituted for the pitch-class integers 10 and 11, respectively. In addition, the input alias values 'T' (ten) and 'E' (eleven) are transformed to 'A' and 'B'. Encodings using mixed alphanumeric values are often better suited to pattern-matching and searching tasks. Encodings using purely numeric values are especially useful when the representation is to be processed numerically.

The **-t** option ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

Note that **pc** will round-off frequencies and non-integer semitone input values to the nearest pitch-class. Hence, an input `**semits` data token of 5.6 will be rendered in the `**pc` output as the value 6. Similarly, the input `**freq` value 452.1 will be rendered in the `**pc` output as the value 9.

In the default operation, **pc** outputs non-pitch-related signifiers in addition to the pitch-class value. For example, the `**semits` token "X15yz" will result in the output "X3yz" — that is, after translating 15 to pitch-class 3, the "Xyz" signifiers are retained in the output. The **-x** option is useful for eliminating non-pitch-class-related signifiers from the output.

EXAMPLES

The following example illustrates the use of **pc.** The input contains four spines — one of which (`**foo`) cannot be processed by **pc.**

```
!! 'pc' example.
**semits      **kern     **pc          **foo
*M2/4         *M2/4      *             *
=1            =1         =1            .
8x            8ee-       abc9 xyz10    A
.             .          BCD           A
#18@          8ff        .             B
23.1 -16      8dd-       .             B
(-2)          8d-        8 7.          C
-12...        .          0.8           C
=2            =2         =2            D
[3.0abc19     [4a-       (2) 3         D
&]            4a-]       6&?           E
=3            =3         =3            E
r             2r         5 4           .
===           ==         ====          .
*-            *-         *-            *-
```

Executing the command

```
pc -xt input > output.pc
```

produces the following result:

```
!! 'pc' example.
**pc        **pc        **pc        **foo
*M2/4       *M2/4       *           *
=1          =1          =1          .
8           3           9 10        A
.           .           B           A
6           5           .           B
11 8        1           .           B
10          1           8 7         C
0           .           0           C
=2          =2          =2          D
3           8           2 3         D
.           .           6           E
=3          =3          =3          E
r           r           5 4         .
===         ==          ====        .
*-          *-          *-          *-
```

Both processed and unprocessed spines are output. Notice that the `**pc` value 'B' in the token 'BCD' has been maintained in the first measure. Notice that for the data tokens in the first spine of measure 2, only the first numerical value in the input data tokens is processed. More than one numerical value is output only if the input token is truly a multiple-stop (as in the third spine of measure 1). Also notice that the tied note in the second spine at the beginning of measure 2 has been rendered as a single value '8' (due to the **-t** option).

FILES

The file *x_option.awk* is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), **cents** (4), ****freq** (2), **freq** (4), ****iv** (2), **iv** (4), ****kern** (2), **kern** (4), ****nf** (2), **nf** (4), ****pc** (2), ****pcset** (2), **pcset** (4), ****pf** (2), **pf** (4), ****pitch** (2), **pitch** (4), ****semits** (2), **semits** (4), ****solfg** (2), **solfg** (4), ****specC** (2) **specC** (4), ****Tonh** (2), **tonh** (4)

NAME

pcset — convert pitch and pitch-class information to set-theoretic representations

SYNOPSIS

pcset [**-c**] [**-n**|**p**|**v**] [*inputfile* ...]

DESCRIPTION

The **pcset** command is used to generate and convert between various set-theoretic representations.

By default, the output is the Fortean pitch-class set type (`**pcset`). Alternatively, the user may choose to output the corresponding *normal form* (`**nf`) or the more succinct *prime form* (`**pf`) or the associated *interval vector* (`**iv`). See REFERENCES below.

In addition to accepting `**semits` or `**pc` inputs, **pcset** can also process `**nf`, `**iv`, `**pf` or `**pcset` inputs. This permits the user to determine the normal form, prime form or interval vector for a given pc-set, or the interval vector for a given pc-set, prime form, or normal form, etc.

For all of the above translations, **pcset** also provides a *complementation* operator, where output values corresponding to the pitch-class set complement. For example, when the complementation option is invoked, an input consisting of the pitch-classes (0,4,7,10) would produce an output pertinent to the complementary set (1,2,3,5,6,8,9,11). Complementation can be applied to any accepted input type, including normal form, prime form, interval vector, and pc-set. Note that if the input consists of all twelve pitch-classes, **pcset** produces a null token as the set complement.

When provided with `**semits` or `**pc` inputs, **pcset** treats each input record as a set of pitches. Unisons and other pitch-class duplications have no effect on the output. Rests within a set of pitches are ignored; where an input record consists solely of one or more rests, a null-token is output.

The **pcset** command is able to translate any of the representations listed below. For descriptions of the various input representations (including `**pcset`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **pcset** command should be given names with the distinguishing extensions '.pcs', '.nf' '.pf' or '.iv' — depending on the selected option.

`**iv`	interval vector representation
`**nf`	normal form for pitch-class sets
`**pc`	pitch-class representation
`**pcset`	Fortean pitch-class set name
`**pf`	prime form representation
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)

Input representations processed by **pcset**.

OPTIONS

The **pcset** command provides the following options:

- **-c** generate output for set complement
- **-h** displays a help screen summarizing the command syntax
- **-n** output normal form (`**nf`) spine
- **-p** output prime form (`**pf`) spine
- **-v** output interval vector (`**iv`) spine

Options are specified in the command line.

By default, the **pcset** command outputs a `**pcset` representation.

The **-n** option causes **pcset** to output normal form (`**nf`) data instead of `**pcset` data. "Normal form" is a standard way of representing the interval structure of any arbitrary set of pitch-classes.

The **-p** option causes **pcset** to output the prime form (`**pf`) for the input. Prime form is analogous to rearranging notes in a chord so that the spelling is in root position, close position, transposed so that the root of the chord is C. By way of example, any major chord (having any root, in any inversion, with any spelling) will have the normal form: 0,4,7 — namely, a given (basic) pitch, plus a pitch 4 semitones above than the basic pitch, plus a pitch 7 semitones above the basic pitch. (See REFERENCES.)

The **-v** option causes interval vector information (`**iv`) to be output rather than `**pcset` data. All pitch-class sets can be characterized according to the possible interval-classes that can be constructed. The six-element interval-vector specifies the abundance of various interval-classes from 1 semitone to 6 semitones. (See REFERENCES.)

Note that the **iv** command, the **nf** command, and the **pf** command are aliases for `pcset -v`, `pcset -n` and `pcset -p` respectively.

EXAMPLES

The following command outputs the interval vectors for the sets formed by successive sonorities in the input file `webern`:

```
pcset -v webern > webern.iv
```

The following command outputs the Fortean set type for the complement of the sonorities given in the input file `berg`:

```
pcset -c berg > berg.pcs
```

The following command outputs the normal form representation for sets formed from successive sonorities in the input file `boulez`:

```
pcset -n boulez.nf
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

context (4), ****iv** (2), **iv** (4), ****nf** (2), **nf** (4), ****pc** (2), **pc** (4), ****pcset** (2), ****pf** (2), **pf** (4), ****semits** (2), **semits** (4)

REFERENCES

Allen Forte, *The Structure of Atonal Music*. New Haven: Yale University Press, 1973.

John Rahn, *Basic Atonal Theory*. New York: Longman Inc., 1980.

Straus, J. *Introduction to Post-Tonal Theory*. Englewood Cliffs, N.J.: Prentice Hall, 1990.

NAME

perform — play Humdrum `**MIDI` files

SYNOPSIS

perform [**-g**] [**-i** *hex*] [**-t** *n.n*] [**-v** *n*] [*inputfile.hmd*]

DESCRIPTION

The **perform** command allows the user to listen to synthesized performances of Humdrum `**MIDI`-format input. When invoked, **perform** provides an interactive environment suitable for proof-listening and other audition tasks.

The **perform** command accepts as input Humdrum inputs containing `**MIDI` representations. The command generates serial MIDI data which are sent directly to a MIDI controller card. The user must have an appropriate MIDI synthesizer connected via a Roland MPU-401 (or compatible) MIDI controller card. All `**MIDI` spines present in the input stream are performed. Non-MIDI spines are ignored and do not affect the sound output.

When first invoked, the **perform** command reads in the entire input into memory. By loading the complete input into memory, random data access is possible and so the user can move freely both forward and backward through the MIDI score. (Note that the size of any input is limited by the available memory.) While the data is being loaded into memory, a tally of the number of bytes loaded is reported. Once the data is loaded, MIDI performance commences immediately.

The **perform** command provides a set of interactive commands that allow the user to pause and resume playback, to change tempo, to move to any measure by absolute or relative reference, and to search forward or backward for commented markers. Editing of MIDI data is not available using **perform.**

Playback can be suspended by typing the space-bar; typing any key will resume playback. By itself, typing ENTER will return to the beginning of the score and re-initiate playback. If a number is typed before pressing ENTER, **perform** will search for a corresponding measure number and continue playback from the beginning of the specified measure. Where more than one measure matches the inputted measure-number, **perform** will find the nearest matching measure, without going backward through "measure zero." Typing a *number* followed by the plus sign (+) will move to the numbered measure corresponding to the current measure plus <*number*>, and continue playback from the beginning of that measure. Typing a *number* followed by the minus sign (-) will move to the numbered measure corresponding to the current measure minus <*number*>, and continue playback from the beginning of that measure. The **perform** commands for moving forward and backward should be avoided when inputs contain unnumbered measures or highly unusual orderings —

such as reverse-order measure numbers.

The tempo can be modified interactively by using the greater-than (>) and less-than (<) signs to increase or decrease the tempo respectively. The less-than sign reduces the tempo whereas the greater-than sign increases the tempo. Tempo changes are in increments of 6 quarter-note durations per minute. A minimum tempo of 8, and a maximum tempo of 250 quarter-notes per minute are permitted. The current tempo is displayed whenever a change of tempo is made.

In the default operation, **perform** echoes all global comments on the screen as the comments are encountered in the input. For inputs containing appropriate annotations, the echoing of comments can provide useful visual markers or reminders of particular moments in the sound output. Whether or not global comments are echoed on the standard output, users can use the **perform** forward-search (/) and backward-search (?) commands to move directly to a particular commented point in the score. For example, if an input contains a global comment containing the character string "Second theme," the user can move immediately to this position in the input by entering the following command:

```
/Second theme
```

followed by a carriage return <cr> (or ENTER). (The search string need only contain sufficient characters to distinguish uniquely the appropriate point of interest.) Similarly, the user can search backward for this character string by entering:

```
?Second theme
```

If the search is successful, playback continues immediately from the new score position. If the search is unsuccessful, playback continues from the current score position.

In rare circumstances, ciphers (stuck notes) can occur in MIDI tasks — such as where an intermittent MIDI cable fails to convey a "note-off" instruction to an active synthesizer. The **p** command ("panic") turns off all active notes. In addition, a "power panic" command (upper-case letter **P**) is provided; this command sends "all-notes-off" commands on MIDI channels 1-16. See also the **tacet** command.

The **f** command will flag a current measure by echoing the measure number on the screen. This can be useful for chronicling various events without having to pause or interrupt playback.

The **perform** command is terminated by typing either **q** or **Q**, or by pressing the escape key (ESC).

The following table summarizes the interactive commands provided by **perform.** A summary of these commands can be displayed from within **perform** by typing the **h** (help) command.

<space>	pause; suspend playback; strike any key to continue
<	reduce tempo
>	increase tempo
–	go back to the beginning of the previous measure and continue performing
integer-	go back *integer* measures and continue performing from the beginning of that measure
+	go forward to the beginning of the next measure and continue performing
integer+	go forward *integer* measures and continue performing from the beginning of that measure
<cr>	by itself the carriage return moves to the beginning of the score and continues performing
/*string*<cr>	search forward from the current position for the next global comment containing *string* and continue performing
?*string*<cr>	search backward from the current position for the nearest global comment containing *string* and continue performing
integer<cr>	go to numbered measure *integer*; where more than one measure shares the same numerical label go to the next (forward) measure matching *integer*
h	display command summary help screen
p	panic; turn off all active notes
p	power panic; reset all notes off on all MIDI channels
q	same as **q**
Q	terminate the perform command
<ESC>	same as **q**

Interactive commands in **perform.**

In order to facilitate user interaction, a number of keyboard 'aliases' are provided. For example, the comma (,) and the period (.) are valid aliases for the less-than (<) and greater-than (>) signs; these aliases normally share the same keys on ASCII keyboards and so avoid the need to use the shift-key. For similar reasons, the equals-sign (=) and the underscore (_) are valid aliases for the plus and minus signs respectively.

OPTIONS

The **perform** command provides the following command-line options:

-g	suppress the echoing of global comments on the standard output
-h	displays a help screen summarizing the command syntax
-i *hex*	assign MIDI input/output address to *hex*
-t *n.n*	set initial tempo to *n.n* times the default tempo
-v *n*	specify default MIDI key-velocity value (0-127)

Options are specified in the command line.

In addition to modifying the tempo interactively while performing, the performance tempo may be specified either in the command line or in the input Humdrum representation. The tempo may be specified on the command line by using the **-t** option. The **-t** must be followed by an integer or real value between 0.13 and 3.80. A value of 1.0 corresponds to the default tempo of 66 quarter-notes per minute. A value of 2.0 doubles the tempo, whereas a value of 0.5 halves the tempo. Alternatively, tempo may be specified using the `**MIDI` tandem interpretation for metronome marking (e.g. `*MM96`). Tempos found in the input representation take precedence over any tempo specified on the command line. If no tempo information is available, **perform** uses the default tempo of 66 quarter-notes per minute.

The **-v** option allows the user to specify a key-velocity default. MIDI instruments normally treat key-velocity data as dynamic or accent information — thus higher key-velocity values are associated with accented notes. Permissible key-velocity values range between 0 and 127. The **-v** option can be used to set the default key-velocity for key-on commands with unspecified key-velocities. In the absence of the **-v** option, the default key-velocity value is 64.

In normal operation, **perform** echoes all global comments on the standard output. This feature may be defeated by invoking the **-g** option.

The **-i** option is used to specify the input/output address of the MIDI card. The default address is '330.' The address is specified as a hexadecimal number.

WARNINGS

When using Microsoft *Windows,* the **perform** command requires the use of "standard mode;" **perform** is unable to work in "386 enhanced mode."

Improper termination of **perform** may leave the MIDI card active, and so possibly to hang the machine. Proper termination of **perform** is achieved via the letters **q** or **Q**, or via the escape key (ESC). In the event of improper termination, the **midreset** command should be used.

PORTABILITY

DOS 2.0 and up. Microsoft *Windows,* in "standard mode" only.

PROPOSED MODIFICATIONS

The program should be modified to allow inputs to contain MIDI control codes and MIDI system exclusive codes.

SEE ALSO

encode (4), **encode.rc** (5), ****MIDI** (2), **midi** (4), **midreset** (4), **record** (4), **smf** (4), **tacet** (4)

REFERENCES

Use of the Music Quest Inc. MIDI library functions is gratefully acknowledged.

NAME

pf — determine prime form for successive vertical sonorities

SYNOPSIS

pf [*inputfile* ...] [> *outputfile.pf*]

DESCRIPTION

The **pf** command is used to determine the *prime form* for any of five set-theory related inputs: pitch (`**semits`), pitch-class (`**pc`), normal form (`**nf`), interval-vector (`**iv`), or Fortean set name (`**pcset`).

"Prime form" is a standardized way of representing the symmetrical interval structure for any arbitrary set of pitch-classes. Prime form is the most intervalically compact representation of a pitch-class set, transposed to begin on pitch-class 0 — where inversions are deemed equivalent. By way of example, any major or minor chord, having any root, in any inversion, consisting of any number of notes, with any spelling, will have the prime form: 0,3,7. See REFERENCES below.

When provided with `**semits` or `**pc` inputs, **pf** treats each input record as a set of pitches. Unisons and other pitch-class duplications have no effect on the output. Rests within a set of pitches are ignored; where an input record consists solely of one or more rests, a null-token is output. The **pf** command can also accept other set theoretic inputs, such as `**nf`, `**iv`, `**pcset`, as well as `**pf` itself.

The following table identifies the input representations accepted by **pf.** For descriptions of the various input representations (including `**pf`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **pf** command should be given names with the distinguishing '.pf' extension.

`**iv`	interval vector representation
`**nf`	normal form representation
`**pc`	pitch-class representation
`**pcset`	Fortean pitch-class set name
`**pf`	prime form representation
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)

Input representations processed by **pf**.

OPTIONS

The **pf** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

The following command outputs the prime form for the sets formed by successive sonorities in the input file `opus24`. The input may be pitches, pitch-classes, Fortean set names, etc.

```
pf opus24 > opus24.pf
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

context (4), ****iv** (2), **iv** (4), ****nf** (2), **nf** (4), ****pc** (2), **pc** (4), ****pcset** (2), **pcset** (4), ****pf** (2), ****semits** (2), **semits** (4)

NOTE

The **pf** command is a shell script that invokes `pcset -p`.

REFERENCES

Allen Forte, *The Structure of Atonal Music*. New Haven: Yale University Press, 1973.

John Rahn, *Basic Atonal Theory*. New York: Longman Inc., 1980.

Straus, J. *Introduction to Post-Tonal Theory*. Englewood Cliffs, N.J.: Prentice Hall, 1990.

NAME

pitch — translate pitch-related representations to American standard pitch notation

SYNOPSIS

pitch [**-tx**] [*inputfile* ...] [> *outputfile.pit*]

DESCRIPTION

The **pitch** command transforms various pitch-related inputs to the corresponding pitch designations approved by the American National Standards Institute (ANSI). The **pitch** command outputs one or more Humdrum `**pitch` spines. ANSI pitch designations use the upper-case letters A to G followed by an optional accidental, followed by an octave number. In the Humdrum `**pitch` representation, optional cents deviation from equal temperament can also be encoded.

The **pitch** command is able to translate any of the pitch-related input representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**pitch`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **pitch** command should be given names with the distinguishing '.pit' extension.

`**cents`	hundredths of a semitone with respect to middle C=0
`**degree`	key-related scale degree
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **pitch**.

OPTIONS

The **pitch** command provides the following options:

-h displays a help screen summarizing the command syntax
-t suppresses printing of all but the first of a group of tied `**kern` notes
-x suppresses printing of non-pitch-related signifiers

Options are specified in the command line.

The **-t** option ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **pitch** outputs non-pitch-related signifiers in addition to the `**pitch` pitch value. For example, the `**Tonh` token "Ges5zzz" will result in the output "Gb5zzz" — that is, after translating Ges5 to Gb5, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "8aa#"; after translating 'aa#' to A#5, the non-pitch-related signifier '8' will also be output, hence the value 8A#5 — which may cause confusion; commands such as **tonh, solfg,** and **pitch** treat the first number encountered in an input token as the octave designation. Hence further processing of this token may lead to it's interpretation as A#8 — or even A#58 — rather than A#5.

The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **pitch.** The input contains six pitch-related spines — two of which (`**deg` and `**cocho`) cannot be processed by **pitch.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'pitch' example.
**kern  **freq       **MIDI       **deg   **metpos  **cocho    **degree  **embell
*M2/4   *M2/4        *M2/4        *M2/4   *M2/4     *M2/4      *M2/4     *M2/4
*       *            *            *       *         *          *d:       *
*       *            *            *       *tb8      *          *         *
=1      =1           =1           =1      =1        =1         =1        =1
8ee-    93foo        /60/bar      1foo    1         r          1/4       ct
.       .            /-60/        .       .         .          .         .
8ff     220          /62/         2       3         9.89       2/4       upt
.       .            /-62/        .       .         .          .         .
8dd-    936.2        /70/         1       2         7.07       3+/4      ct
.       .            /-70/        .       .         .          .         .
8d-     277.18       /61/         6       3         7.135      7/3       sus
.       .            /-61/        .       .         .          .         .
=2      =2           =2           =2      =2        =2         =2        =2
[4a-    r            .            5       1         r          r         .
.       .            .            7       3         5.5        1/4       ct
4a-]    300          /48/ /52/    1       2         8.11       6+/4      ct
.       .            /-48/        .       .         .          .         .
.       82.4 261.6   /-52/        2       3         7.33 6.4   3/4 5/4   ct
=3      =3           =3           =3      =3        =3         =3        =3
r       512          .            r       1         r          3/4 1/5   .
==      ==           ==           ==      ==        ==         ==        ==
*-      *-           *-           *-      *-        *-         *-        *-
```

Executing the command

```
pitch -tx input > output.pit
```

produces the following result:

```
!! 'pitch' example.
**pitch  **pitch  **pitch  **deg   **metpos  **cocho   **pitch  **embell
*M2/4    *M2/4    *M2/4    *M2/4   *M2/4     *M2/4     *M2/4    *M2/4
*        *        *        *       *         *         *d:      *
*        *        *        *       *tb8      *         *        *
=1       =1       =1       =1      =1        =1        =1       =1
Eb5      F#2+9    C4       1foo    1         r         D4       ct
.        .        .        .       .         .         .        .
F5       A3       D4       2       3         9.89      E4       upt
.        .        .        .       .         .         .        .
Db5      Bb5+7    Bb4      1       2         7.07      F#4      ct
.        .        .        .       .         .         .        .
Db4      Db4      Db4      6       3         7.135     C#3      sus
.        .        .        .       .         .         .        .
=2       =2       =2       =2      =2        =2        =2       =2
Ab4      r        .        5       1         r         r        .
.        .        .        7       3         5.5       D4       ct
.        D4+36    C3 E3    1       2         8.11      B4       ct
.        .        .        .       .         .         .        .
.        E2 C4    .        2       3         7.33 6.4  F4 A4    ct
=3       =3       =3       =3      =3        =3        =3       =3
r        C5-37    .        r       1         r         F4 D5    .
==       ==       ==       ==      ==        ==        ==       ==
*-       *-       *-       *-      *-        *-        *-       *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**freq`, `**MIDI`, and `**cocho` spines have been stripped away (due to the **-x** option). In the case of the `**degree` input, **pitch** recognizes the spelling of various pitches in the context of the key of D minor. Hence, the raised third degree is F#, and the raised sixth degree is B natural. Also note the presence of cents-deviation from equal temperament in the translation of the `**freq` data (second spine).

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), **cents** (4), ****degree** (2), **degree** (4), ****freq** (2), **freq** (4), ****fret** (2), **hint** (4), ****kern** (2), **kern** (4), ****MIDI** (2), **midi** (4), **mint** (4), ****pitch** (2),

****semits** (2), **semits** (4), ****solfg** (2), **solfg** (4), ****specC** (2) **specC** (4), ****Tonh** (2), **tonh** (4)

NAME

proof — check syntax of `**kern` file

SYNOPSIS

proof [-w] [*inputfile.krn* ...]

DESCRIPTION

The **proof** command checks the syntax of `**kern`-encoded files. It may be used to assist in the detection and correction of `**kern` encoding errors.

The **proof** command scans the input stream for ten types of errors: (1) consistency of incremented measure numbers, and measure-number agreement between concurrent parts, (2) incoherent pitch tokens (such as assigning two pitches to the same note), (3) incoherent durations (such as assigning two durations to the same note), (4) notes that have simultaneous sharp and flat accidentals, (5) repeat encodings of trills, mordents, inverted mordents, ties, pauses, staccato markings, and natural signs, (6) consistent phrase markings (e.g. '{' must precede and be matched with '}' etc. with the exception of elisions (&)), (7) consistent slur markings (e.g. '(' must precede and be matched with ')' etc. with the exception of elisions (&)), (8) tied notes which differ in pitch or pitch spelling, (9) unequal total durations for each voice within a measure, and (10) the presence of multiple-stops that do not share the same duration. In addition, **proof** issues a number of warnings of possible data errors: (11) warn of files which do not end with a double barline, (12) warn of material encoded following a double barline, (13) warn of measures which change meters unannounced, (14) warn of the presence of identical material in consecutive measures, (15) warn of the presence of pitches which are encoded without an accidental — where the same pitch had previously received an accidental in the same measure, and (16) warn of the absence of meter, tempo, key, and key signature declarations.

OPTIONS

The **proof** command provides the following options:

- **-h** displays a help screen summarizing the command syntax
- **-w** suppress output of warnings

Options are specified in the command line.

The **-w** option causes all warnings to be suppressed; only `**kern` error messages are output.

DIAGNOSTICS

The following list tabulates all of the potential errors and warnings issued by the **proof** command.

```
Error: Incoherent key signature, line n: assigned more than one accidental to
       the same pitch.
Error: Inconsistent barline indication across spines at line n.
Error: Timing error in measure x, spine y, at line n
Error: Double stops at line x in spine n do not have same duration.
Error: Unmatched ties in spine n.
Error: Incorrect pitch specification in spine x, line n
Error: Incorrect duration specification in spine x, line n
Error: Incorrect accidental specification in spine x, line n
Error: Incorrect ornament specification in spine x, line n
Error: Tie must begin on valid kern note, spine n
Error: Incorrect tie encoding in spine x, line n
Error: Tied notes inconsistent in spine x, line n
Error: Incorrect tie specification in spine x, line n
Error: Incorrect tie in spine x, line n
Error: Incorrect slur, spine x, line n
Error: Incorrect slur specification in spine x, line n
Error: Incorrect phrase marking in spine x, line n
Error: Incorrect phrase specification in spine x, line n
Error: Only one pause permitted in data token. Spine x, line n
Warning: No double barline in input.
Warning: No meter declaration for spine n.
Warning: No tempo declaration for spine n.
Warning: No key declaration for spine n.
Warning: No key signature declaration for spine n.
Warning: Measure n may be out of place near line n
Warning: Material follows after double barline at line n
Warning: Measure j identical to measure k at line n
Warning: Possible change of meter in measure x, line n
Warning: Accidental may be missing in m.x, spine y, at line n
Warning: No double barline in input.
```

Potential errors and warnings issued by **proof.**

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

census (4), **encode** (4), **humdrum** (4), ****kern** (2)

NAME

recode — recode numeric tokens in selected Humdrum spines

SYNOPSIS

recode -f *reassign-file* **-i** '***interpretation*' [**-s** *regexp*] [**-x**] [*inputfile* ...]

DESCRIPTION

The **recode** command is used to recode numeric components of data tokens in selected input spines. Typically, **recode** is used to reassign a range of numerical values into a finite set of classes or categories. For example, **recode** could be used to reassign all numerical values less than zero to the value -1, and to assign all values greater than or equal to zero to the value +1. A typical use of **recode** might be to reassign melodic intervals (represented in semitones) to one of five categories: (1) unison [0 semits], (2) up step [plus 1 or 2 semits], (3) up leap [plus 3 or more semits], (4) down step [minus 1 or 2 semits], (5) down leap [minus 3 or more semits]. Similarly, duration information might be rhythmically "justified" so that all durations near 0.5 seconds are recoded as precisely 0.5 seconds.

Note that **recode** will modify only those input spines matching the exclusive interpretation specified in the command line.

The manner by which numeric values are reassigned is specified by the user in a separate *reassignment file.* Reassignment files consist of one or more reassignment records; each record specifies a *condition* and a resulting replacement *string.* When the condition is satisfied, the numerical data is replaced by the associated string. A simple reassignment file is:

```
==0	zero
!=0	other
```

This file contains two reassignment records. *Conditions* are given in the left column and the associated replacement *strings* are given in the right column. Conditions and strings are separated by a single tab. Given the above reassignment, when a numerical value in an input token is equal to zero, the output replaces the input number by the alphabetic string "`zero`." The second condition (!= means not-equals) indicates that if a numerical value not equal to zero is encountered in an input token, the output replaces the number by the alphabetic string "`other`."

Permissible relational operators are listed in the following table.

==	equals
!=	not equals
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
else	default relation

Relational operators for **recode**

Permissible replacement strings include any combination of printable ASCII characters with the exception of the tab.

Conditions are tested in the order given in the reassignment file. Thus if a numeric value satisfies more than one condition, only the first string replacement is made. Consider, for example, the following reassignment file:

```
<=0     LOW
>100    HIGH
>0      MEDIUM
```

In this case, all numeric values are replaced by one of three strings: `HIGH`, `MEDIUM`, or `LOW`. The order of specification is important in the above file. If the `MEDIUM` condition was specified prior to the `HIGH` condition, then all values greater than one hundred would be categorized as `MEDIUM` rather than as `HIGH`.

The *else* relation can be used to specify the default string output for numeric input values that satisfy none of the preceding conditions in the reassignment file. If no *else* condition is specified and none of the other conditions are satisfied, **recode** outputs the original input token without any modification.

Substitutions are made even when a number is embedded in non-numeric data. For example, given the above reassignment file, an input token `foo200bar` would be output as `fooHIGHbar`. That is, the numeric portion of the input string (200) would be deemed to satisfy the condition (>100) and so would be replaced by the string ("HIGH").

An important property of the **recode** command is that string replacements are limited to the **first** occurrence of numeric data within each data token. Subsequent numeric data within the token remains untouched. Thus, using the above reassignment file, the input token `foo200bar300` would be output as `fooHIGHbar300`.

In the case of multiple-stops (data tokens having two or more parts separated by spaces), **recode** processes the first occurrence of numeric data for each part of the token. For example, the double-stop token `foo200 bar300` would be output as `fooHIGH barHIGH`.

The **recode** command provides options to identify which data tokens may be excluded (skipped) in processing (**-s**), plus an option that suppresses the echoing of unprocessed signifiers in the output (**-x**). See OPTIONS for further information.

OPTIONS

The **recode** command provides the following options:

-f *reassign*	use reassignments given in file *reassign*
-h	displays a help screen summarizing the command syntax
-i '***interp*'	process only ***interp* spines
-s *regexp*	skip; completely ignore tokens matching *regexp*; (echo in output only)
-x	(exclude) do not echo unprocessed data signifiers in the output

Options are specified in the command line.

The user can suppress the echoing of non-numeric data within a token by specifying the **-x** option on the command line. When this exclude option is selected, only the replacement strings are output. For example, given the following reassignment file:

```
<=0     LOW
>100    HIGH
>0      MEDIUM
```

The input token `foo200bar` would be output as `HIGH`. If a data token contains no numeric component, then the **-x** option causes a null token to be output.

The **-x** option also suppresses the echoing of unprocessed numerical components. (Recall that string replacements made by **recode** are limited to the first occurrence of numerica data within a data token.) For example, with the **-x** option, the input data token `foo200bar17` would be output as `HIGH`.

Processing of certain types of data tokens may be avoided by invoking the **-s** (skip) option. This option must be accompanied by a user-define regular-expression (see **regexp (6)**). Input data tokens matching this expression are not processed and are simply echoed in the output. This option may be useful, for example, in avoiding the processing of barlines, or other types of data.

EXAMPLES

The operation of the **recode** command can be illustrated by referring to the following hypothetical Humdrum file named `patrie`.

```
**kern	**abc
16g	0
8.g	00
16g	1
=1	=1
4cc	2.0
4cc	+3.
4ee	4
4ee	-1
=2	=2
4.gg	22.
8ee	1.1
8.cc	.1
16cc	x1X
8.ee	x1x2x
16cc	1 2
=3	=3
4a	.
4r	r
*-	*-
```

Consider also the following "reassignment" file, named `reassign`.

```
==0	zero
==1	one
==2	two
<0	negative
<=3	<=3
>4	>4
else	???
```

The command:

```
recode -s = -i '**abc' -f reassign patrie
```

would produce the following output:

```
**kern    **abc
16g       zero
8.g       zero
16g       one
=1        =1
4cc       two
4cc       <=3
4ee       ???
4ee       negative
=2        =2
4.gg      >4
8ee       <=3
8.cc      <=3
16cc      xoneX
8.ee      xonex2x
16cc      one two
=3        =3
4a        .
4r        r
*-        *-
```

Notice the following: (1) the measure numbers `1` and `2` have remained unchanged due to the skip option `-s =`, (2) the input `x1X` has been replaced by the output string `xoneX` (non-numeric data remain in the same relative position), (3) the input `x1x2x` has been replaced by the output string `xonex2x` (only the first numerical value in each token is modified), (4) the double-stop input `1 2` has been replaced by the output string `one two`, and (5) both the null token (.) and the non-numeric token (`r`) have been echoed in the input unchanged.

Note that with the **-x** option, all of the non-numeric signifiers in `**abc` spine would be suppressed in the output. The single non-numeric token (`r`) would be output as a null token.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

humsed (4), **rend** (4), **regexp** (4), **regexp** (6), **sed** (UNIX)

NAME

record — record live MIDI input in Humdrum `**MIDI` data format

SYNOPSIS

record [**-i** *hex*] [**-q** *n*] [> *outputfile.hmd*]

DESCRIPTION

The **record** command captures a stream of input MIDI data and translates this data into a simple Humdrum `**MIDI` representation. Input MIDI data is obtained through a Roland MPU-401 (or compatible) interface — usually connected in turn to a MIDI synthesizer. The obtained `**MIDI` data can be manipulated using several Humdrum tools, or it can be played-back using the **perform** command.

Recording commences as soon as the command is invoked. Recording ceases when any ASCII key is pressed — with the exception of the space bar. Only MIDI key-press activity (including after-touch) information is recorded. MIDI system-exclusive instructions and other non-key-press data are not recorded.

Each MIDI channel is represented using a separate Humdrum spine. New spines are added automatically during the recording — in response to additional activity on new MIDI channels. Once a MIDI channel becomes active, the corresponding Humdrum spine continues to be output until the recording is terminated.

At any time during the recording process, pressing the space bar will insert a `**MIDI` barline data token in the output stream. Measure numbers are incremented automatically beginning with measure 1.

It is recommended that output files produced using the **record** command should be given names with the distinguishing '.hmd' extension.

OPTIONS

The **record** command provides the following option:

-h	displays a help screen summarizing the command syntax
-i *hex*	assign MIDI interface input/output address to *hex*
-q *n*	invokes quantizing using a temporal window of *n* clock ticks

Options are specified in the command line.

The **-q** option invokes a quantizing function where timing information is rounded-off to a specified level of resolution. This option may be used to eliminate expressive timing information and assist in producing a canonical duration representation. The degree of quantizing is specified by the *n* argument to the **-q** option, where *n* represents the quantizing window in MIDI clock ticks. Recorded events occurring within this window are deemed to be simultaneous, and are recorded as Humdrum double-stops in the output.

The **-i** option is used to specify the input/output address of the MIDI card. The default address is '330.' The address is specified as a hexadecimal number.

SAMPLE OUTPUT

The following examples illustrate how **record** may be used. A simple command invocation is:

```
record
```

Output `**MIDI` data may appears as follows:

```
!! Data from the MPU-401 MIDI card.
**MIDI
*Ch1
236/67/64
12/-67/64
10/67/66
11/-67/64
13/67/51
12/-67/64
14/63/72
263/-63/64
84/65/61
15/-65/64
10/65/55
15/-65/64
11/65/51
23/-65/64
12/62/58
171/-62/64
*-
```

Using the quantizing option:

```
record -q 10
```

might produce output such as the following `**MIDI` data. Notice the frequent occurrence of multiple-stops (more than one note-instruction in the spine).

```
!! Data from the MPU-401 MIDI card.
!! Quantizing set at 10 clock ticks.
**MIDI
*Ch1
303/50/39
13/-50/64 13/74/55
23/76/43
15/-74/64 15/78/58 15/-76/64
22/69/35 22/-78/64 22/62/43
18/-62/64 18/78/43 18/-69/64
22/76/35
14/-78/64 14/74/58
15/-76/64 15/-74/64
12/81/48 12/54/77
17/-54/64 17/74/69 17/-81/64
23/76/48
19/78/66 19/-74/64 19/-76/64
21/62/43 21/69/69 21/-78/64
14/-62/64 14/78/51 14/-69/64
25/76/58
17/-78/64 17/74/74 17/-76/64
15/-74/64
*-
```

DIAGNOSTICS

The program is implemented as a four-state finite state machine.

PORTABILITY

DOS 2.0 and up, with a Roland MPU-401 or compatible MIDI interface.

SEE ALSO

cents (4), **encode** (4), **encode.rc** (5), **kern** (4), ****MIDI** (2), **midi** (4), **perform** (4), **pitch** (4), **semits** (4), **smf** (4) **solfg** (4), **tonh** (4)

REFERENCES

Use of the Music Quest Inc. MIDI library functions is gratefully acknowledged.

NAME

regexp — interactive regular-expression tester

SYNOPSIS

regexp [*inputfile*]

DESCRIPTION

The **regexp** command invokes an interactive pattern-matcher that is useful for formulating and refining regular expressions. *Regular expressions* provide a generic means for defining patterns of characters (see tutorial in Section 6). Innumerable UNIX and Humdrum commands make use of regular expressions. The **regexp** command allows the user to test interactively various expressions using a sample text. If no sample text is supplied by the user *(inputfile)* then a short default text is used.

Once invoked, the user may interactively input a regular expression followed by a carriage return. The sample text is scanned for occurrences of the defined regular expression. Any text lines containing the matched expression are displayed on the screen; **regexp** differs from the UNIX **grep** command in that the precise locations of the matched pattern are explicitly marked. (See EXAMPLES below.) Note that only the first occurrence of a matching pattern is identified in each line of text. (This is how most software tools make use of regular expressions.)

The entire sample text file may be viewed by typing the regular expression `.*` or by simply typing a carriage return. Viewing the sample text is helpful in identifying character-strings that are not identified by a given regular expression.

The **regexp** command is terminated by typing an end-of-file marker (control-D on UNIX; control-Z on DOS or OS/2).

The **regexp** command implements the same regular expression features found in the UNIX **awk** command. This includes all so-called "extended" regular expression features with the exception of \> and \<.

OPTIONS

The **regexp** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

Imagine the case where the sample text file specified in the command line contains the following three records:

```
The quick brown fox jumped over the lazy dogs.
Once upon a time, long, long ago ...
It was the best of times, it was the worst of times.
```

The following regular expression defines any character string beginning with the lower-case letter 'b', followed by zero or one instance of any single character, followed by a lower-case vowel.

```
b.?[aeiou]
```

Given this regular expression, the corresponding output would appear as follows:

```
The quick brown fox jumped over the lazy dogs.
          ^ ^
It was the best of times, it was the worst of times.
           ^^
```

Notice that only those text lines matching the defined regular expression are displayed in the output.

WARNINGS

The regular-expression features provided by **regexp** depend on the local UNIX **awk** utility — as accessed via the `AWK_VER` shell variable. Available features may change depending on the version of **awk** used.

FILES

The default text file is `$HUMDRUM/regexp.txt`.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

awk (UNIX), **expr** (UNIX), **grep** (UNIX), **gres** (UNIX), **patt** (4), **pattern** (4), **regexp** (6), **sed** (UNIX)

NAME

reihe — output specified row variant for a given prime row

SYNOPSIS

reihe -[**I**|**i**] *n* [**-a**] [**-m** *n*] *primefile*
reihe -[**P**|**p**] *n* [**-a**] [**-m** *n*] *primefile*
reihe -[**R**|**r**] *n* [**-a**] [**-m** *n*] *primefile*
reihe -[**R**|**r**] *primefile*
reihe -[**RI**|**ri**] *n* [**-a**] [**-m** *n*] *primefile*
reihe -[S|s] *n* *primefile*

DESCRIPTION

The **reihe** command outputs a specified row variant for a given prime row input. Normally, **reihe** is used to generate tone-row variants for 12-tone rows. However, the "tone-rows" can consist of any number of pitch-classes and need not be based on modulo-12 class-equivalence. In certain circumstances the "prime row" may consist of non-numeric or non-pitch-related data — such as articulation marks or dynamic marks. In addition to the traditional set-theoretic transformations, **reihe** also permits cyclic rotation (shifting) of set elements.

The input to the **reihe** command is interpreted as a *prime* or basic set of elements from which a transformed set is to be derived and output. Four traditional types of set-transformations can be generated by specifying the appropriate option: **-P** (prime), **-I** (inversion), **-R** (retrograde), and **-RI** (retrograde-inversion). Either upper- or lower-case characters can be used when specifying these options.

Following each option is a numerical parameter. For prime and inversion forms, the numerical parameter indicates the pitch-class value for the first "note" of the output form. For example, **-P 0** specifies the prime form beginning on pitch-class 0; **-P 1** specifies the prime transposed so that it begins on pitch-class 1. For retrograde and retrograde-inversion forms, the numerical parameter specifies the *last* value of the output row.

Positive integers greater than 11 are permitted in the input — but are treated as modulo-12 equivalent (unless the **-m** option specifies a different modulus). In the case of `**pc` inputs, the upper-case letters A and B are accepted as aliases for pitch-classes 10 and 11 respectively. Following the Fortean practice, pitch-classes 10 and 11 may alternatively be represented in the input by the upper-case letters T (ten) and E (eleven) — although this latter convention is discouraged.

In addition to the traditional set-transformations of transposition, inversion, retrograde, and retrograde-inversion, **reihe** also provides for rotation — where the set elements are cyclically *shifted* by a specified number of positions. For example, **-s 3** causes each set element to be shifted forward by 3 positions. The shift transformation can be combined with each of the other traditional transformations only by invoking the **reihe** command twice in succession.

Normally, **reihe** is used to transform numerical data (typically pitch-class values). However, the retrograde (**-r**) and shift (**-s**) operations can be performed on any data (including non-numeric data — such as articulation marks). For non-numeric data, the retrograde option must be invoked without a numerical parameter. Attempting to transpose non-numeric data will result in an error.

The prime-form input file must contain a single spine. For untransposed retrogrades and rotational shift transformations, any single-spine Humdrum input will be accepted. If the **-m** option is invoked, any input interpretation will be accepted provided all data tokens contain numbers only. In the case of tranposed prime, inversion, and retrograde transformations, the input must conform to the Humdrum pitch-class (`**pc`) representation. In all cases, comments, null tokens, and tandem interpretations in the input spine are ignored and are not echoed in the output. In the case of `**pc` inputs, barlines and rests are also ignored. Output interpretations always echo the input interpretation.

By way of example, consider the following input file `webern`:

```
!! Anton von Webern
!! Klavierstueck, opus posthumous
**pc
=1
9
10
.
11
8
7
=2
1
2
3
r
6
5
4
=3
0
*-
```

Executing the command:

```
reihe -p 1 webern
```

produces the following output:

```
**pc
1
2
3
0
11
5
6
7
10
9
8
4
*-
```

Notice that the comments, barlines, rests, and null tokens have been eliminated from the input file. This leaves the output in a form better suited to pattern matching using the **patt** or **pattern** commands.

Similarly, executing the command:

```
reihe -r 1 webern
```

produces:

```
**pc
1
5
6
7
4
3
2
8
9
0
11
10
*-
```

OPTIONS

The **reihe** command provides the following options:

-a	for `**pc` inputs, output alphanumeric representation (where A=10, B=11)
-h	displays a help screen summarizing the command syntax
-I *n*	output inversion set-form starting on pitch-class *n*
-i *n*	same as **-I** option
-m *n*	calculate according to modulo *n* arithmetic
-P *n*	output prime set-form starting on pitch-class *n*
-p *n*	same as **-P** option
-R	output retrograde of input row
-R *n*	output retrograde set-form ending on pitch-class *n*
-r	same as **-R** option
-r *n*	same as **-R** *n* option
-RI *n*	output retrograde-inversion set-form ending on pitch-class *n*
-ri *n*	same as **-RI** option
-S [±]*n*	output set-form shifted *n* elements forward (+) or backward (-)
-s [±]*n*	same as **-S** option

Options are specified in the command line.

When the **-a** option is invoked, pitch-class inputs (`**pc`) will produce pitch-class outputs using the alias values 'A' for pitch-class 10, and 'B' for pitch-class 11. (See the ****pc** representation.)

By default, the **reihe** command assumes modulo 12 arithmetic for prime, inversion, retrograde, and retrograde-inversion transformations. In other words, transposing the numerical value '11' up three pitch-classes results in an output value of '2.' The **-m** option can be used to specify some other modulo value. If this option is invoked with `**pc` input, the alias values (A=T=10; B=E=11) are disabled and only numerical data can be processed and output. The **-m** and **-a** options are thus mutually exclusive.

EXAMPLES

The sample document given below shows a 5-tone row used in Igor Stravinsky's "Dirge-Canons" from *In Memoriam Dylan Thomas*.

```
!! I. Stravinsky, 5-tone row
**pc
2
3
6
5
4
*-
```

The command: `reihe -s -1 igor` would result in the following output:

```
**pc
3
6
5
4
2
*-
```

The command: `reihe -a -i 2 igor` would result in the following output:

```
**pc
2
1
A
B
0
*-
```

The command: `reihe -ri 0 -m 7 igor` would result in the following output:

```
**pc
5
4
3
6
0
*-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****iv** (2), **iv** (4), ****nf** (2), **nf** (4), **patt** (4), **pattern** (4), ****pc** (2), **pc** (4), ****pcset** (2), **pcset** (4), ****pf** (2), **pf** (4), **recode** (4), **semits** (4)

NAME

rend — split data tokens from specified spines into component sub-tokens

SYNOPSIS

rend [-s] **-i** '*target_interp*' **-f** *reassign-file* [*inputfile* ...]

DESCRIPTION

The **rend** command breaks apart data tokens from selected input spines into one or more sub-tokens distributed across one or more newly created output spines. The user specifies which input spine or spines are to be split. The manner in which the signifiers are to be distributed is specified in a separate *reassignment* file.

Humdrum data tokens often contain more than one type of information or type of signfier. For example, the `**pitch` representation consists of three parts:† the pitch letter name, the accidental, and the octave number (e.g. A#4). In some tasks it may be useful to split such information into separate spines. For example, a user may wish to reformat the following spine:

```
**pitch
Ab3
Eb4
F#4
C5
*-
```

as three independent spines:

```
**octave	**note	**accidental
3	Ab	b
4	Eb	b
4	F#	#
5	C	.
*-	*-	*-
```

The **rend** command allows each occurrence of a target exclusive interpretation to be replaced by specified output spines. The user selects how the signfiers (characters) in the input spines are to be distributed to the replacement output spines. Signifiers (ASCII characters) are identified using UNIX regular expression syntax (see **regexp (6)**).

† Not including cents deviation.

The above transformation may be achieved by invoking the following command:

```
rend -i '**pitch' -f reassign
```

The **-i** option specifies the target input interpretation, i.e., the input spine(s) to be processed. The **-f** option specifies a reassignment-file (named `reassign`) containing the following records:

```
**octave        [0-9]
**note          [A-Gb#x]+
**accidental    [b#x]+
```

Reassignment files consist of one or more records, each containing two strings separated by a tab. The left-most string identifies the name of the new spine to be generated. The right-most string defines an associated regular expression. Any input signifiers matching the regular expression will be echoed as output in the associated spine. In the above case, all numbers are echoed in the first spine (`**octave`), all letters plus the sharp (#) and flat (b) signs are echoed in the second spine (`**note`), whereas only sharp and flat signs are echoed in the third spine (`**accidental`). The order of the output spines preserves the order of the assignments in the reassignment file. In the above case, for example, the order of the output spines will be `**octave`, `**note`, `**accidental` for each input spine labelled `**pitch`.

OPTIONS

The **rend** command provides the following options:

-f *reassign*	maps input tokens to output tokens according to definitions given in the file *reassign*
-h	displays a help screen summarizing the command syntax
-i *target_interp*	process all input spines whose exclusive interpretations are labelled *target_interp*
-s	matches a single instance of the given pattern rather than all instances

Options are specified in the command line.

EXAMPLES

Consider the following example:

```
rend -i '**kern' -f noterest song01
```

and the associate reassignment file named `noterest`:

```
**notes     []A-Ga-g[#-]+|^=+[0-9]*
**rests     [\.0-9r]+|^=+[0-9]*
```

This command specifies that each `**kern` spine in the file `song01` is to be split into two new spines dubbed `**notes` and `**rests`. The first regular expression — `'[]A-Ga-`

`g[#-]+|^=+[0-9]*'` — indicates that the following strings should be echoed in the data records for `**notes`: the upper-case letters `A` to `G` and the lower-case letters `a` to `g`, plus the characters `[`, `]`, `#`, and `-`. Alternatively, **rend** will echo any data token beginning with one or more equals-signs, followed by zero or more numbers.

Similarly, the second `**rests` spine will contain characters that match the regular expression '[\.0-9r]+|^+=[0-9]*'. This includes the period (.), all numbers (0-9), plus the letter *r*. Alternatively, **rend** will echo any data token beginning with one or more equals-signs, followed by zero or more numbers.

Given this command, the following input:

```
**kern    **lyrics   **kern
!! Commented input.
8.G       Hi-        4r
16G#      de-        .
=23       =23        =23
8A        ho-        2r
[8c       .          .
8c]       .          .
16r       .          .
16A       .          .
=24       =24        =24
2C 2E     hum.       2r
===       ===        ===
*-        *-         *-
```

will produce the following output:

```
**notes   **rests   **lyrics   **notes   **rests
!! Commented input.
G         8.        Hi-        .         4r
G#        16        de-        .         .
=23       =23       =23        =23       =23
A         8         ho-        .         2r
[c        8         .          .         .
c]        8         .          .         .
.         16r       .          .         .
A         16        .          .         .
=24       =24       =24        =24       =24
C E       2 2       hum.       .         2r
===       ===       ===        ===       ===
*-        *-        *-         *-        *-
```

Notice that **rend** correctly handles Humdrum multiple-stops (such as `2C 2E`). Notice also that if no match is made, a null token (.) is output.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

cleave (4), **extract** (4), **humsed** (4), **recode** (4), **regexp** (4), **regexp** (6)

WARNINGS

Note that, apart from spine-path terminators, no other spine-path indicators are permitted in spines containing the target interpretation.

BUGS

If the interpretation targetted for processing appears in a spine starting with a different interpretation, the output will fail to generate the proper spine terminator and add-spine path indicators. The result is a non-Humdrum file. Consider the following command:

```
rend -i '**ex1' -f reassign input
```

and the associate reassignment file (`reassign`):

```
**let  [a-z]
**num  [0-9]
```

Given the following input:

```
**ex1    **ex2
a1       b2
*tand1   *tand2
c3       d4
*        **ex1
e5       f6
*-       *-
```

the corresponding output is given below. Note the absence of appropriate spine-path indicators between lines 4 and 5 (hence the output is non-Humdrum).

```
**let    **num    **ex2
a        1        b2
*tand1   *tand1   *tand2
c        3        d4
*        *        **let    **num
e        5        f        6
*-       *-       *-       *-
```

NAME

rid — eliminate specified record types

SYNOPSIS

rid [**-dgiltuDGILU**] [*inputfile* ...]

DESCRIPTION

The **rid** command allows the user to eliminate specified types of Humdrum records (lines) from the input stream. Depending on the options selected, **rid** will eliminate global comments, local comments, interpretations, duplicate exclusive interpretations, tandem interpretations, data records, data records consisting of just null tokens (null data records), empty global or local comments, empty interpretations, or any combination of these record types.

Humdrum comments are records that begin with an exclamation mark (!). Local comments begin each active spine with a single exclamation mark, whereas gobal comments begin with two exclamation marks (!!) at the beginning of the record. A global comment consisting of only two exclamation marks is a null global comment. If a local comment record contains only single exclamation marks in each spine, it is dubbed a null local comment.

Humdrum interpretations are records that begin with an asterisk (*). Tandem interpretations begin each active spine with a single asterisk, whereas exclusive interpretations begin with an asterisk and have at least one active spine beginning with two asterisks. If each active spine contains only a single asterisk, the record is dubbed a null interpretation.

Records that are not comments or interpretations are deemed to be data records. Null tokens are data tokens consisting of just the period character (.). A data record containing only null tokens (separated by tabs) is a null data record.

A duplicate exclusive interpretation is an exclusive interpretation that repeats the name of an interpretation that is already active for a given spine. If a spine has not been terminated, there is frequently little need to indicate (again) the active interpretation for a given spine. (An exception occurs when the user wants the data to be processed as discontinuous — such as avoiding calculating pitch intervals between the last note of one piece and the first note of a subsequent piece.)

OPTIONS

The **rid** command provides the following options:

-h	displays a help screen summarizing the command syntax
-D	remove all data records
-d	remove null data records
-G	remove all global comments
-g	remove null global comments
-I	remove all interpretation records
-i	remove null interpretation records
-L	remove all local comments
-l	remove null local comments
-T	remove all tandem interpretations
-U	remove unnecessary (duplicate) exclusive interpretations
-u	same as **-U**

Options are specified in the command line.

In general, upper-case options eliminate *all* records of a given type, whereas the corresponding lower-case options eliminate only null records (devoid of signifiers) for a given record type.

EXAMPLES

The following examples illustrate the use of **rid.** Consider the following input file:

```
!! 'rid' example
!!
**abc	**xyz
*tand	*em
12	.
.	.
!local	!comments
*	*
*x	*x
**xyz	**abc
!	!
.	34
*-	*-
```

The following command:

```
rid -dlu input
```

will eliminate all null data records, all null local comments, and any unnecessary (duplicate) exclusive interpretations:

```
!! 'rid' example
!!
**abc    **xyz
*tand    *em
12       .
!local   !comments
*        *
*x       *x
.        34
*-       *-
```

Alternatively, the command:

```
rid -DGLiT input
```

will eliminate all data records, all global and local comments, all null interpretations, and all tandem interpretations:

```
**abc   **xyz
*x      *x
**xyz   **abc
*-      *-
```

WARNINGS

Removing Humdrum interpretation records will render the file inconsistent with the Humdrum syntax. Further processing of such a file with the Humdrum tools may be impossible.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

humsed (4), **recode** (4), **sed** (UNIX)

NAME

scramble — randomize order of either Humdrum data records or data tokens

SYNOPSIS

scramble **-r** [-s *regexp*] [*inputfile*] **scramble** **-t** [**-m**] [-s *regexp*] [*inputfile*]

DESCRIPTION

The **scramble** command can be used to randomize either the order of Humdrum data records or the arrangement of data tokens within each data record. The **scramble** command is useful for generating control data when testing contextual relationships between signifiers.

Two modes of operation are supported according to whether the **-r** (record) or **-t** (token) option is invoked. When the **-r** mode is selected, the order of data records is randomized. In this mode, each output record is identical to some input record; only the order of the output records is changed. When the **-t** mode is selected, the order of data tokens within each record is randomized. In this mode, the order of the input records is preserved — however, data tokens between concurrent spines are randomly swapped. The **-t** mode will also cause sub-tokens within multiple stops to be rearranged within the data token. However, if the **-m** option is concurrently invoked, then sub-tokens within multiple-stops will be randomly redistributed across all tokens in the record. The **-r** and **-t** options cannot be invoked concurrently.

In both modes of operation, Humdrum comments and interpretations remain unaffected. Comments and interpretations are output intact, and in precisely the same location (line number) as in the input. Only data records are affected by **scramble.**

Each time **scramble** is invoked, a different random ordering is generated.

Note that when using the **-r** mode, the scrambling process may produce an output that is no longer syntactically correct Humdrum. With the **-r** mode, **scramble** is guaranteed to produce Humdrum output only if (1) the input file is bona fide Humdrum, and (2) the number of spines in the input does not vary.

With the **-t** mode, **scramble** will always produce output conforming to the Humdrum syntax, however it can produce uninterpretable output if concurrent spines do not contain the same type of data (i.e. have the same exclusive interpretations).

Notice also that reordering data records may destroy data-token links such as `**kern` ties.

A skip option (-s) permits users to anchor certain data records so that they are not either repositioned, or their data tokens rearranged.

OPTIONS

The **scramble** command provides the following option:

-h	displays a help screen summarizing the command syntax
-m	redistribute subtokens in multiple stops across all tokens in the record; used in conjunction with **-t** only
-r	scramble the order of data records; don't scramble data tokens
-s *regexp*	skip; don't scramble records matching *regexp*; leave matching records intact, and in the same position
-t	scramble data tokens within each record; don't scramble record order

Options are specified in the command line. One of either the *record mode* (**-r**) or *token mode* (**-t**) must be invoked.

EXAMPLES

The use of the **scramble** command can be illustrated using the following input:

```
!! A global comment
!! Another comment
**inter    **inter
*abcd      *efgh
=1         =1
1a 1b      a
!local     !local
2          b1 b2 b3
3          c
=2         =2
!! A later comment.
4a 4b      d
=3         =3
5          e
*-         *-
```

When processed using the *record mode*, the command:

```
scramble -r -s = inputfile
```

might produce the following output:

```
!! A global comment
!! Another comment
**inter     **inter
*abcd       *efgh
=1          =1
3           c
!local      !local
5           e
1a 1b       a
=2          =2
!! A later comment.
4a 4b       d
=3          =3
2           b1 b2 b3
*-          *-
```

In this example, notice that the Humdrum comments and interpretations remain in their original location; only the data records have been reordered. In addition, data records containing an equals-sign have been frozen in their original locations.

When processed using the *token mode,* the command:

```
scramble -t -m -s = inputfile
```

might produce the following output:

```
!! A global comment
!! Another comment
**inter      **inter
*abcd        *efgh
=1           =1
1b           a 1a
!local       !local
b2           b3 2 b1
c            3
=2           =2
!! A later comment.
4a 4b        d
=3           =3
5            e
*-           *-
```

Notice that a complete scrambling of data tokens within a Humdrum file cannot be achieved merely by invoking one **scramble** mode followed by the other mode. In order to completely scramble a Humdrum file the user must extract and scramble the record order for each spine independently, and then reassemble the scrambled spines into a new file using the **assemble** command.

Note also that where the number of spines changes over the course of the input file, valid

Humdrum output is unlikely. Outputs consistent with the Humdrum syntax can be ensured by using the **fields -i** command to chronicle changing numbers of spines in a file, followed by the **yank** command to segregate data blocks containing the same number of spines. Each such bock can be **scrambled** independently and then the blocks reconnected using the UNIX **cat** command. Unnecessary (duplicate) interpretations can be eliminated using **rid -u.**

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

assemble (4), **extract** (4), **fields** (4), **humdrum** (4)

WARNINGS

If the number of spines changes over the course of the input file, valid Humdrum output is unlikely when using the **-r** option. Note also that reordering data records or data tokens will destroy data-token links such as `**kern` "ties." Use of the **-t** mode, can produce uninterpretable output when concurrent spines do not contain the same interpretations.

NAME

semits — translate pitch-related representations to numerical semitones

SYNOPSIS

semits [**-p** *n*] [**-tx**] [*inputfile* ...] [> *outputfile.sem*]

DESCRIPTION

The **semits** command transforms various pitch-related inputs to corresponding numerical semitone values. It outputs one or more Humdrum `**semits` spines containing values corresponding to the semitone distance from middle C for pitch-related input tokens. Pitches above middle C produce positive output values, whereas pitches below middle C produce negative output values. For example, the `**pitch` token "C3" is transformed to –12 (semits).

The **semits** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**semits`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **semits** command should be given names with the distinguishing '.sem' extension.

`**cents`	hundredths of a semitone with respect to middle C=0 (e.g. 1200 equals C5)
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **semits**.

OPTIONS

The **semits** command provides the following options:

-h displays a help screen summarizing the command syntax
-p *n* output precision of *n* decimal places
-t suppresses printing of all but the first note of a group of tied `**kern` notes
-x suppresses printing of non-semits data

Options are specified in the command line.

The **-p** option can be used to set the precision of the output values to *n* decimal places. The default precision is integer values only. Note that **semits** is able to process `**semits` as input; this feature allows the user to round-off existing `**semits` data to a specified precision.

The **-t** ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **semits** outputs non-pitch-related signifiers in addition to the semits value. For example, the `**pitch` token "A5zzz" will result in the output "21zzz" — that is, after translating A5 to 21 semits, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "8aa"; after translating 'aa' to 21 semits, the non-pitch-related signifier '8' will also be output, hence the value 821 — which will undoubtedly cause confusion. The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **semits.** The input contains six pitch-related spines — two of which (`**deg` and `**cocho`) cannot be processed by **semits.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'semits' example.
**kern  **pitch   **MIDI     **deg   **metpos  **cocho    **Tonh   **embell
*M2/4   *M2/4     *M2/4      *M2/4   *M2/4     *M2/4      *M2/4    *M2/4
*       *         *          *       *tb8      *          *        *
=1      =1        =1         =1      =1        =1         =1       =1
8ee-    G#4foo    /60/bar    1foo    1         r          Gis2     ct
.       .         /-60/      .       .         .          .        .
8ff     A3        /62/       2       3         9.89       H2       upt
.       .         /-62/      .       .         .          .        .
8dd-    Ab3       /70/       1       2         7.07       B2       ct
.       .         /-70/      .       .         .          .        .
8d-     C#4       /61/       6       3         7.135      Cis4     sus
.       .         /-61/      .       .         .          .        .
=2      =2        =2         =2      =2        =2         =2       =2
[4a-    r         .          5       1         r          r        .
.       .         .          7       3         5.5        Heses2   ct
4a-]    D4        /48/ /52/  1       2         8.11       C3       ct
.       .         /-48/      .       .         .          .        .
.       D4 F4     /-52/      2       3         7.33 6.4   C3 Es3   ct
=3      =3        =3         =3      =3        =3         =3       =3
r       G4        .          r       1         r          H2 D3    .
==      ==        ==         ==      ==        ==         ==       ==
*-      *-        *-         *-      *-        *-         *-       *-
```

Executing the command

```
semits -tx input > output
```

produces the following result:

```
!! 'semits' example.
**semits  **semits  **semits  **deg   **metpos  **cocho   **semits  **embell
*M2/4     *M2/4     *M2/4     *M2/4   *M2/4     *M2/4     *M2/4     *M2/4
*         *         *         *       *tb8      *         *         *
=1        =1        =1        =1      =1        =1        =1        =1
15        8         0         1foo    1         r         -16       ct
.         .         .         .       .         .         .         .
17        -3        2         2       3         9.89      -13       upt
.         .         .         .       .         .         .         .
13        -4        10        1       2         7.07      -14       ct
.         .         .         .       .         .         .         .
1         1         1         6       3         7.135     1         sus
.         .         .         .       .         .         .         .
=2        =2        =2        =2      =2        =2        =2        =2
8         r         .         5       1         r         r         .
.         .         .         7       3         5.5       -15       ct
.         2         -12 -8    1       2         8.11      -12       ct
.         .         .         .       .         .         .         .
.         2 5       .         2       3         7.33 6.4  -12 -9    ct
=3        =3        =3        =3      =3        =3        =3        =3
r         7         .         r       1         r         -13 -10   .
==        ==        ==        ==      ==        ==        ==        ==
*-        *-        *-        *-      *-        *-        *-        *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**pitch`, `**MIDI`, and `**cocho` spines have been stripped away (due to the **-x** option).

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), **cents** (4), ****freq** (2), **freq** (4), ****fret** (2), ****kern** (2), **kern** (4), ****MIDI** (2), **midi** (4), ****pitch** (2), **pitch** (4), ****semits** (2), ****solfg** (2), **solfg** (4), ****specC** (2) **specC** (4), ****Tonh** (2), **tonh** (4)

NAME

simil — measure the non-numeric similarity between two single-spine inputs

SYNOPSIS

simil [**-nr**] [**-x** *length*] *sourcefile* [*templatefile*] [> *outputfile.sim*]

DESCRIPTION

The **simil** command measures the degree of similarity between two single-spine inputs. Similarity measures are calculated by determining the minimum *edit distance* — that is, the least amount of editorial manipulation required to transform the two inputs so that they are identical. In contrast to the **correl** command, **simil** can act on both numeric and non-numeric data, and so can be used to characterize similarity for any form of Humdrum input.

Two inputs are required by **simil** — the *source* and *template* inputs. Both inputs must contain single columns of data; multi-column inputs are forbidden. The *source* input must conform to the Humdrum syntax, however the *template* should contain only data records. Comments and interpretations should not appear in the *template.* In addition, following the initial exclusive interpretation, all comments and interpretations in the *source* input should be removed.

Depending on the mode of operation, **simil** outputs either one or two spines of continuous information regarding the similarity of the two inputs. The length of **simil**'s output matches that of the *source* file.

Two modes of operation are provided by **simil**: the *fixed template mode* and the *variable template mode.* The fixed template mode is useful for scanning a source input for patterns similar to a given template. This mode of operation is useful when the user knows in advance what feature or pattern is being sought. The variable template mode, by contrast, provides an automated means for discovering common patterns shared by the template and source inputs.

In the *fixed template mode,* a single output spine is generated, dubbed `**simil`. Output similarity values are numbers ranging between zero and one. These numbers indicate the relative similarity between the source and template inputs at the current location in the *source* file. Values near zero indicate great dissimilarity whereas a value of precisely 1.00 indicates that the template and source match exactly at the current position.

In *fixed template mode,* the entire *template* input is treated as a single pattern and compared with the *source* input. In this mode, the template may not be longer than the source input. If the length of the template is the same as the length of the source input, then only a single output value is generated — representing the edit-distance similarity of the two files. The remaining output is padded by null tokens. If the template is shorter than the source input, then the source input is scanned using the template. For each data record in the source input the edit-distance similarity is measured between the entire template and a string of corresponding length beginning at the current point in the source input. Each successive output value indicates the degree of similarity between the template and source inputs as the template is shifted along the source input.

In the *variable template mode,* the template input provides a "reservoire" from which multiple templates are derived. Specifically, the template input is broken-up into a series of (shorter) overlapping "subordinate" templates. The length of each of these subordinate templates is determined by the **-x** option parameter. For example, consider a template input consisting of the values: 1, 2, 3, 4 — each number appearing on successive lines. With **-x 3**, two subordinate templates will be generated, each consisting of three data records: 1, 2, 3 and 2, 3, 4. With **-x 2**, three subordinate templates will be generated, each consisting of two data records: 1, 2; 2, 3; and 3, 4.

Two output spines are generated in the *variable template mode:* `**simil` and `**simxrf`. The `**simxrf` spine provides cross-reference information identifying the place in the template input where a subordinate pattern is highly similar to the current position in the source file. As each record is encountered in the source input, **simil** scans the list of all possible subordinate templates and identifies the template with the highest similarity value. This value is output (in the `**simil` spine) along with the *line number* in the original template input where the subordinate template begins. If more than one subordinate template shows the same similarity value, then the line numbers for each high-similarity template appear in the `**simxrf` spine, separated by commas. Common subordinate patterns will appear frequently in the `**simxrf` output.

It is recommended that output files produced using the **simil** command should be given names with the distinguishing '.sim' extension.

FURTHER DETAILS

The **simil** program implements the Damerau-Levenshtein metric for edit distance (see REFERENCES). Permissible edit operations include substitutions and deletions. Each edit action incurs a penalty, and the cummulative edit-distance penalty determines the similarity.

In the default operation, **simil** assigns equivalent edit penalties (1) for deletions and substitutions. However, the user can explicitly define these penalties.

The **simil** command allows the user to define the cost of each edit operation via an initialization file. The initialization file must be named `simil.rc` and be located in the current directory or the user's home directory. Arbitrary costs may be assigned to any of eight edit operations shown in the following table:

Name Tag	Edit Operation
D1	Delete a nonrepeated token in String 1
D2	Delete a nonrepeated token in String 2
R1	Delete a repeated token in String 1
R2	Delete a repeated token in String 2
S0	Substitute a token that is repeated in neither String 1 nor String 2
S1	Substitute a token that is repeated in String 1 only
S2	Substitute a token that is repeated in String 2 only
S3	Substitute a token that is repeated in String 1 and String 2

Edit operations used by **simil**.

In describing the edit operations, String 1 is the source string and String 2 is the template string. Notice that there is no overt edit operation for insertion: an insertion in String 1 is deemed equivalent to a deletion in String 2. However, different edit penalties may be defined for deletions from String 1 (D1) compared with deletions from String 2 (D2). In musical applications defining such asymmetrical penalties may be important. For example, two inputs may represent a basic melody and an embellished variant of the melody. Using asymmetrical penalties allows the user to specify that the deletion of tones from the embellished version is less costly than deletion of tones from the basic melody.

Since repetition is a common form of musical variation, **simil** allows the user to distinguish between repeated and non-repeated tokens. A repeated token is defined as one that is immediately preceded by an identical token. Thus, in deleting a sequence of identical symbols in String 1, say, all deletions except the first occurrence are R1 operations, whereas the deletion of the first occurrence is a D1 operation.

Note that the minimum theoretical edit-distance for any set of penalty weightings can be determined empirically by providing the **simil** program with source and template strings that share no symbols in common. For example, the source input may consist entirely of numbers, whereas the template input consists entirely of alphabetic characters. In the case where all edit operations are assigned a penalty of +1, the minimum quantitative similarity between two strings is 0.37.

Some user-defined weightings may give rise to peculiar results — such as negative costs — but **simil** does not forbid this. **Simil** generates warning messages if the weighting seem illogical; for example, if the cost of R1 is more than that of D1. In addition, **simil** will abort operation if the defined edit penalties transgress the triangular inequality (see REFERENCES). The default weighting for all operations is +1.

Below is a sample initialization file that defines the R1 substitution has having an edit penalty of 0.7, whereas the R2 substitution is given a penalty of 0.9. Edit penalties are defined by specifying the operation, followed by some spaces or tabs, followed by some real number. Since no other penalties are defined in this file, the remaining edit operations use the default edit penalty of 1.0. If any operation is assigned more than one weight, the latest assignment is used. The user may effectively eliminate a given edit operation by defining an arbitrarily high edit penalty.

```
# This is a comment.
R1      0.7
R2      0.9
```

OPTIONS

The **simil** command provides the following options.

-n	do not scale similarity measures according to template length
-r	reverse the order of *source* and *template* inputs on the command line; permits the source file to be entered using the standard input.
-x *length*	invoke *variable template mode*; break-up template file input into subordinate patterns of length *length*

Options are specified in the command line.

Raw edit-distance scores are normally unreliable estimates of similarity, unless the length of the template is considered. For example, 3 editing operations constitutes a rather modest change for a template consisting of 20 elements. However, 3 edit operations is significant for a template consisting of only 5 elements. In the default operation, **simil** scales the edit-distance scores according to the length of the comparison template. This ensures that all similarity values remain between 0 and 1. The **-n** option defeats this scaling procedure, and outputs the raw similarity scores.

The **-r** option reverses the order of the *source* and *template* input specifications on the command line. If both inputs are files, this option is of little use. Where one input is to be typed manually via the standard input, this option allows the user to specify a template file as input, and to type the source document manually.

The **-x** option invokes the *variable template mode* discussed above. The numerical argument given to the **-x** option determines the length of the subordinate templates drawn from the template file.

EXAMPLES

The following examples illustrate the operation of **simil.** Consider first, the *fixed template mode*. In the following example, the source input consists of the left-most spine (labelled `**foo`) and is held in a file named `source`; the middle column (not Humdrum) consists of the letters A, B and C, and is held in the file named `template`. The following command:

```
simil source template
```

generates the third column (labelled `**simil`):

```
(source   (template   (simil
 input)     input)    output)
**foo     A           **simil
X         B           0.51
A         C           1.00
B                     0.51
C                     0.37
D                     0.51
A                     0.72
B                     0.72
B                     0.51
C                     0.51
B                     .
A                     .
*-                    *-
```

Each successive value in the output spine is matched with a data token in the source input file. For example, the second value (1.00) in the `**simil` spine arises from an exact match of the (A, B, C) pattern *beginning* with the second data token in the source input. The second highest value (0.72) occurs in both the sixth and seventh `**simil` data records, indicating that fairly similar sequences occur beginning with the sixth and seventh data records in the source input. Specifically, **simil** has recognized that the sequence (A, B, B, C) is only one edit-operation (a deletion) different from the template (A, B, C). In the ensuing record, **simil** has recognized that the sequence (B, B, C) is only one edit-operation (substitution A/B) different from (A, B, C). Notice that the final value (0.51) indicates that the edit distance for (C, B, A) is less like the template. Also notice that the lowest value (0.37) corresponds to an input pattern (beginning D, D, A) that bears little resemblance to the template.

If the above input were pitches, it might be argued that *changing* a pitch is more dissimilar than *repeating* a pitch. In the following `simil.rc` file, an increased penalty has been assigned for dissimilar substitution, and decreased penalties have been assigned for repetition.

```
S0   1.6
S1   0.7
S3   0.7
```

Repeating the above command with this new `simil.rc` file produces the following results:

```
(source    (template    (simil
 input)     input)      output)
**foo      A            **simil
X          B            0.51
A          C            1.00
B                       0.51
C                       0.26
D                       0.51
A                       0.79
B                       0.59
B                       0.51
C                       0.51
B                       .
A                       .
*-                      *-
```

Notice that the similarity measure for the pattern (A, B, B, C) has increased from 0.72 to 0.79, whereas the similarity measure for (B, B, C) has decreased from 0.72 to 0.59.

Consider now the use of the *variable template mode.* Once again, we will use the same source and template files. Given the short length of the template, there is little choice regarding the length of the subordinate templates. In the following command, a template length of two elements is specified:

```
simil -x 2 source template
```

This command produces the following output:

```
*simil   *simxrf
0.37     1,2
1.00     1
1.00     2
0.37     1,2
0.37     1,2
1.00     1
0.61     1,2
1.00     2
0.61     1
0.61     2
0.61     1
.        .
```

Only two two-element subordinate patterns are possible given out template — A, B and B, C. The first subordinate template begins on line 1 of the template file, while the second subordinate template begins on line 2. The `**simxrf` spine identifies which of the subordinate patterns is most similar to the source file at the given input record. The `**simil` spine identifies the corresponding similarity measure for the most similar pattern. For example, the second and third `**simil` records both report similarity values of 1.00. However, the first instance is associated with the pattern beginning on template record 1 (A, B), whereas the second instance is associated with the pattern (B, C) beginning on template record 2.

PORTABILITY

DOS 2.0 and up. OS/2. All UNIX systems.

SEE ALSO

context (4), ****correl** (2), **correl** (4), **patt** (4), **pattern** (4)

WARNINGS

In *variable template mode,* execution times may be quite lengthy.

REFERENCES

The Damerau-Levenshtein metric is described in P. Hall & G. Dowling "Approximate string matching," *ACM Computing Surveys,* Vol. 12 (1980) pp. 381-402.

The theory and operation of *simil* is explained in Keith Orpen and David Huron, "Measurement of similarity in music: A quantitative approach for non-parametric representations," *Computers in Music Research,* Vol. 4 (1992), pp. 1-44.

AUTHOR

Written by Keith S. Orpen, University of Waterloo. Copyright 1992.

NAME

smf — generate standard MIDI file from Humdrum `**MIDI` input

SYNOPSIS

smf [**-t** *n.n*] [**-v** *n*] *inputfile.hmd* > *outputfile.smf*

DESCRIPTION

The **smf** command allows the user to create "standard MIDI files" from Humdrum `**MIDI`-format files. Standard MIDI files (SMFs) are industry-standard binary files that can be imported by a variety of MIDI applications software packages, including sequencer programs and several music notation packages.

The **smf** command accepts as input Humdrum files containing `**MIDI` representations. All `**MIDI` spines present in the input stream are translated to SMF MIDI. Non-`**MIDI` spines are ignored and will not affect the SMF output.

OPTIONS

The **smf** command provides the following command-line options:

-h	displays a help screen summarizing the command syntax
-t *n.n*	set initial tempo to *n.n* times the default tempo
-v *n*	specify default MIDI key-velocity value (0-127)

Options are specified in the command line.

The performance tempo may be specified either in the command line or in the input Humdrum representation. The tempo may be specified on the command line by using the **-t** option. The **-t** must be followed by an integer or real value between 0.13 and 3.80. A value of 1.0 corresponds to the default tempo of 66 quarter-notes per minute. A value of 2.0 doubles the tempo, whereas a value of 0.5 halves the tempo. Alternatively, tempo may be specified using the `**MIDI` tandem interpretation for metronome marking (e.g. `*MM96`). Minimum and maximum tempi are 8 and 250 quarter-notes per minute respectively. Tempo specifications found in the input representation take precedence over any tempo specified on the command line. If no tempo information is available, **smf** assumes a default tempo of 66 quarter-notes per minute.

The **-v** option allows the user to specify a key-velocity default. MIDI instruments normally treat key-velocity data as dynamic or accent information — thus higher key-velocity values are associated with accented notes. Permissible key-velocity values range between 0 and 127. The **-v** option can be used to set the default key-velocity for key-on commands with unspecified key-velocities. In the absence of the **-v** option, the default key-velocity value is 64.

DIAGNOSTICS

The **smf** command produces MIDI Standard "format Ø" files. When used in conjunction with the **midi** command, the **midi -d** option should be invoked in order to ensure that all notes are turned off.

PORTABILITY

DOS 2.0 and up. DOS 2.0 and up. OS/2. UNIX.

PROPOSED MODIFICATIONS

The program should be modified to allow inputs to contain MIDI control codes and MIDI system exclusive codes.

SEE ALSO

****MIDI** (2), **midi** (4), **perform** (4), **record** (4)

REFERENCES

Use of the Music Quest Inc. MIDI library functions is gratefully acknowledged.

NAME

solfa — translate pitch-related representations to tonic solfa syllables (`**solfa`)

SYNOPSIS

solfa [**-tx**] [*inputfile* ...] [> *outputfile.sol*]

DESCRIPTION

The **solfa** command transforms various pitch-related inputs to the corresponding tonic solfa syllables. The command outputs one or more Humdrum `**solfa` spines — where pitches are designated by the syllables *do, re, mi, fa, so, la,* and *ti* — or their chromatic alterations: *di, da, ri, ra,* etc. (see below). Tonic solfa syllables can be determined only with reference to some prevailing key. For example, the pitch C is the tonic (`do`) in the key of C major, but the mediant (`mi`) in the key of A-flat major. The **solfa** command expects a tandem interpretation indicating the key of the input passage; **solfa** will adapt to specified changes of key within an input stream. If no key information is provided prior to the first pitch-related **data**, **solfa** issues an error message and terminates.

There are various systems for extending the tonic solfa syllables in order to representing chromatic alterations. The system used by **solfa** is tabulated below. (Pronunciations are indicated in parentheses.)

basic	**raised**	**lowered**
do (*doe*)	di (*dee*)	de (*day*)
re (*ray*)	ri (*ree*)	ra (*raw*)
mi (*me*)	my (*my*)	me (*may*)
fa (*fah*)	fi (*fee*)	fe (*fay*)
so (*so*)	si (*see*)	se (*say*)
la (*la*)	li (*lee*)	le (*lay*)
ti (*tee*)	ty (*tie*)	te (*tay*)

Summary of **solfa** *Signifiers*

The **solfa** command differs from the **deg** and **degree** commands in that pitches are represented without regard to major or minor *mode*. For example, in the key of C major, **deg** and **degree** will characterize A-flat as a lowered sixth scale degree, whereas the same pitch will be a normal sixth scale degree in the key of C minor. In the case of **Bsolfa**, the A-flat will be characterized as `le` — whether or not the key is C major or C minor. As in the case of **deg** and **degree**, the amount of chromatic alteration is not represented; once a pitch is "raised," raising it further will not change the output representation. For example, where the tonic pitch is B-flat, both B-natural and B-sharp are represented by `di`.

The **solfa** command is able to translate any of the pitch-related representations listed below. For descriptions of the various input representations (including `**solfa`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced by the **solfa** command should be given names with the distinguishing .sol extension.

`**kern`	core pitch/duration representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**solfg`	French solfège system (fixed 'doh')
`**Tonh`	German pitch system

Input representations processed by **solfa**.

OPTIONS

The **solfa** command provides the following options:

-h displays a help screen summarizing the command syntax
-t suppresses printing of all but the first note of a group of tied notes
-x suppresses printing of non-`**solfa` signifiers

Options are specified in the command line.

The **-t** option ensures that only a single output value is given for tied notes; the output coincides with the first note of the tie.

In the default operation, **solfa** outputs non-pitch-related signifiers in addition to the degree value. For example, in the key of D, the `**kern` token "4Gz" will result in the output "4faz" — that is, after translating G to fa, the "4...z" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing. The **-x** option is useful for eliminating non-pitch-related signifiers from the output.

EXAMPLES

The following example illustrates the use of **solfa.** The input contains four pitch-related spines — one of which (`**MIDI`) cannot be processed by **solfa.** In addition, there is one non-pitch-related spines (`**embell`).

```
!! 'solfa' example.
**kern	**Tonh	**MIDI	**solfg	**pitch	**embell
*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4
*C:	*d:	*G#:	*a:	*F:	*F:
=1	=1	=1	=1	=1	=1
8ee-	Gis2	/60/	do3	F4foo	ct
.	.	/-60/	.	.	.
8f	H2	/62/	fa3	G4bar	upt
.	.	/-62/	.	.	.
8dd-	B2	/70/	mi3	E4	ct
.	.	/-70/	.	.	.
8d--	Cis4	/61/	r	F4	sus
.	.	/-61/	.	.	.
=2	=2	=2	=2	=2	=2
[4a-	r	.	mi_b3	F4 A4	.
.	Heses2	.	re3	G4 Bb4	ct
4a-]	C3	/48/ /52/	do3	E4 C5	ct
.	.	/-48/	.	.	.
.	H2 E3	/-52/	la3	G4	ct
=3	=3	=3	=3	=3	=3
r	A2 F3	.	r	F4	.
==	==	==	==	==	==
*-	*-	*-	*-	*-	*-
```

Executing the command:

```
solfa -tx input > output
```

produces the following result:

```
!! 'solfa' example.
**solfa    **solfa    **MIDI       **solfa    **solfa    **embell
*M2/4      *M2/4      *M2/4        *M2/4      *M2/4      *M2/4
*C:        *d:        *G#:         *a:        *F:        *F:
=1         =1         =1           =1         =1         =1
me         fi         /60/         me         do         ct
.          .          /-60/        .          .          .
fa         la         /62/         le         r          upt
.          .          /-62/        .          .          .
ra         le         /70/         so         ti         ct
.          .          /-70/        .          .          .
ra         ti         /61/         r          do         sus
.          .          /-61/        .          .          .
=2         =2         =2           =2         =2         =2
le         r          .            so         do mi      .
.          le         .            fa         re fa      ct
.          te         /48/ /52/    me         ti so      ct
.          .          /-48/        .          .          .
.          la re      /-52/        do         re         ct
=3         =3         =3           =3         =3         =3
r          so me      .            r          do         .
==         ==         ==           ==         ==         ==
*-         *-         *-           *-         *-         *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**pitch` spine have been stripped away (due to the **-x** option).

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****deg** (2), **deg** (4), ****degree** (2), **degree** (4), ****kern** (2), **kern** (4), ****pitch** (2), **pitch** (4), ****solfa** (2), ****solfg** (2), **solfg** (4), ****Tonh** (2), **tonh** (4)

NAME

solfg — translate pitch-related representations to French solfège notation

SYNOPSIS

solfg [**-tx**] [*inputfile* ...] [> *outputfile.slg*]

DESCRIPTION

The **solfg** command transforms various pitch-related inputs to the corresponding French system for designating pitches. It outputs one or more Humdrum `**solfg` spines. French pitch designations use the so-called "fixed-doh" system, where: *do, ré, mi, fa, sol, la,* and *si* correspond to *C, D, E, F, G, A,* and *B*. In `**solfg`, flats *(bémol)* and sharps *(dièse)* are abbreviated *b* and *d* respectively. Hence, 'do dièse' (`do~d`) for C sharp, 'la bémol' (`la~b`) for A flat, 'sol double-dièse' (`sol~dd`) for G double-sharp, 'si double-bémol' (`si~bb`) for B double-flat, and so on.

The **solfg** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**solfg`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **solfg** command should be given names with the distinguishing '.slg' extension.

`**cents`	hundredths of a semitone with respect to middle C=0
`**degree`	key-related scale degree
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **solfg**.

OPTIONS

The **solfg** command provides the following options:

- **-h** displays a help screen summarizing the command syntax
- **-t** suppresses printing of all but the first note of a group of tied `**kern` notes
- **-x** suppresses printing of non-solfg data

Options are specified in the command line.

The **-t** ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In the default operation, **solfg** outputs non-pitch-related signifiers in addition to the solfg value. For example, the `**pitch` token "Gb5zzz" will result in the output "sol~b5zzz" — that is, after translating Gb5 to sol~b5, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "8aa#"; after translating 'aa#' to la~d5, the non-pitch-related signifier '8' will also be output, hence the value 8la~d5 — which may cause confusion. Commands such as **pitch** and **solfg** treat the first number encountered in an input token as the octave designation. So further processing of this token may lead to it's interpretation as A#8 — or even A#85 — rather than A#5.

The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **solfg.** The input contains six pitch-related spines — two of which (`**deg` and `**cocho`) cannot be processed by **solfg.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'solfg' example.
**kern  **pitch  **MIDI        **deg  **metpos  **cocho   **degree  **embell
*M2/4   *M2/4    *M2/4         *M2/4  *M2/4     *M2/4     *M2/4     *M2/4
*       *        *             *      *tb8      *         *d:       *
=1      =1       =1            =1     =1        =1        =1        =1
8ee-    G#4foo   /60/bar       1foo   1         r         1/4       ct
.       .        /-60/         .      .         .         .         .
8ff     A3       /62/          2      3         9.89      2/4       upt
.       .        /-62/         .      .         .         .         .
8dd-    Ab3      /70/          1      2         7.07      3+/4      ct
.       .        /-70/         .      .         .         .         .
8d-     C#4      /61/          6      3         7.135     7/3       sus
.       .        /-61/         .      .         .         .         .
=2      =2       =2            =2     =2        =2        =2        =2
[4a-    r        .             5      1         r         r         .
.       .        .             7      3         5.5       1/4       ct
4a-]    D4       /48/ /52/     1      2         8.11      6+/4      ct
.       .        /-48/         .      .         .         .         .
.       D4 F4    /-52/         2      3         7.33 6.4  3/4 5/4   ct
=3      =3       =3            =3     =3        =3        =3        =3
r       G4       .             r      1         r         3/4 1/5   .
==      ==       ==            ==     ==        ==        ==        ==
*-      *-       *-            *-     *-        *-        *-        *-
```

Executing the command

```
solfg -tx input > output
```

produces the following result:

```
!! 'solfg' example.
**solfg  **solfg   **solfg  **deg  **metpos  **cocho   **solfg  **embell
*M2/4    *M2/4     *M2/4    *M2/4  *M2/4     *M2/4     *M2/4    *M2/4
*        *         *        *      *tb8      *         *d:      *
=1       =1        =1       =1     =1        =1        =1       =1
mi~b5    sol~d4    do4      1foo   1         r         re4      ct
.        .         .        .      .         .         .        .
fa5      la3       re4      2      3         9.89      mi4      upt
.        .         .        .      .         .         .        .
re~b5    la~b3     si~b4    1      2         7.07      fa~d4    ct
.        .         .        .      .         .         .        .
re~b4    do~d4     re~b4    6      3         7.135     do~d3    sus
.        .         .        .      .         .         .        .
=2       =2        =2       =2     =2        =2        =2       =2
la~b4    r         .        5      1         r         r        .
.        .         .        7      3         5.5       re4      ct
.        re4       do3 mi3  1      2         8.11      si4      ct
.        .         .        .      .         .         .        .
.        re4 fa4   .        2      3         7.33 6.4  fa4 la4  ct
=3       =3        =3       =3     =3        =3        =3       =3
r        sol4      .        r      1         r         fa4 re5  .
==       ==        ==       ==     ==        ==        ==       ==
*-       *-        *-       *-     *-        *-        *-       *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**pitch`, `**MIDI`, and `**cocho` spines have been stripped away (due to the **-x** option). In the case of the `**degree` input, **solfg** recognizes the spelling of various pitches in the context of the key of D minor. Hence, the raised third degree is `fa~d` (F#), and the raised sixth degree is `si` (B natural).

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), **cents** (4), ****degree** (2), **degree** (4), ****freq** (2), **freq** (4), ****fret** (2), ****kern** (2), **kern** (4), ****MIDI** (2), **midi** (4), **mint** (4), ****pitch** (2), **pitch** (4), ****semits** (2), **semits** (4), ****solfg** (2), ****specC** (2), **specc** (4), ****Tonh** (2), **tonh** (4)

NAME

strophe — selectively extract strophic data

SYNOPSIS

strophe [-s *strophe_list* | **-x** *strophe_label*] [*inputfile* ...]

DESCRIPTION

The **strophe** command is used to isolate or extract selective strophic data. Strophic data encode alternative concurrent information for a given passage. Examples of alternative concurrent information might include the texts for different verses of a song, alternative renditions of the same passage (such as *ossia* passages), or differing editorial interpretations of a given note or sequence of notes.

The **strophe** command permits the user to extract selected information paths (called *strophes*) present in a Humdrum input.

Structurally, strophic data must begin from a single common spine, split apart into two or more alternative spines, and then rejoin to form a single spine. Since the strophes split from a common spine, all strophic data necessarily begin by sharing the same data type. Different exclusive interpretations may be introduced in the strophic passage — provided all strophic spines end up sharing the same data type just prior to being rejoined.

The beginning of a strophic passage is signalled by the presence of a *strophic passage initiator* — a single asterisk followed by the keyword "strophe" (`*strophe`). The end of a strophic passage is signalled by the *strophic passage terminator* — a single asterisk followed by the upper-case letter 'S' followed by a minus sign (`*S-`). Each spine within the strophic passage begins with a *strophe label* and ends with a *strophe end indicator* (`*S/fin`).

Strophe labels may consist of either alphanumeric names, or numbers. Numerical labels should be used when the strophic data imply some sort of order, such as verses in a song. Alphanumeric labels are convenient for distinguishing different editions or *ossia* passages. The following example encodes a melodic phrase containing four numbered verses from "Das Wandern" from *Die Schöne Müllerin* by Schubert.

```
!! Franz Schubert, 'Das Wandern' from "Die Schoene Muellerin"
**kern          **text
*>[V,V,V,V]     *>[V,V,V,V]
*>V             *>V
*k[b-e-]        *Deutsch
*               *solo
*               *strophe
*               *^
*               *^              *^
*               *S/1            *S/2          *S/3        *S/4
8f              Das             Vom           Das         Die
=5              =5              =5            =5          =5
8f              Wan-            Was-          sehn        Stei-
8b-             -dern           -ser          wir         -ne
8a              ist             ha-           auch        selbst ,
8ee-            des             -ben          den         so
=6              =6              =6            =6          =6
(16dd           Mül-            wir's         Rä-         schwer
)16ff           |               |             |           |
(16dd           -lers           ge-           -dern       sie
)16b-           |               |             |           |
8f              Lust ,          -lernt ,      ab ,        sind ,
8dd             das             vom           den         die
=7              =7              =7            =7          =7
(8.cc           Wan-            Was-          Rä-         Stei-
)16a            |               |             |           |
8b-             -dern !         -ser !        -dern !     -ne !
8r              %               %             %           %
*               *S/fin          *S/fin        *S/fin      *S/fin
*               *v              *v            *v          *v
*               *S-
*-              *-
```

Notice that this file contains a single section labelled 'V' (verse) and that an expansion list occurs near the beginning of the file that indicates the section is to be repeated 4 times in total.

The strophic passage in the above example pertains only to the spine marked `**text`. Following the strophic passage indicator (`*strophe`), the spine is split apart until the required number of verses are generated. Then each spine is labelled with its own strophe label. Since the verses have an order, it is appropriate to label them with numbers: `*S/1`, `*S/2`, and so on. The individual verses are terminated with strophe end indicators (`*S/fin`), the spines rejoin, and then a strophic passage terminator (`*S-`) marks the end of the strophic passage.

Executing the command:

```
strophe -s 4
```

produces the following output:

```
!! Franz Schubert, 'Das Wandern' from "Die Schoene Muellerin"
**kern          **text
*>[V,V,V,V]     *>[V,V,V,V]
*>V             *>V
*k[b-e-]        *Deutsch
*               *solo
8f              Die
=5              =5
8f              Stei-
8b-             -ne
8a              selbst ,
8ee-            so
=6              =6
(16dd           schwer
)16ff           |
(16dd           sie
)16b-           |
8f              sind ,
8dd             die
=7              =7
(8.cc           Stei-
)16a            |
8b-             -ne !
8r              %
*-              *-
```

Strophic encodings are nearly always encoded in *abbreviated* rather than *through-composed* file formats. Abbreviated encodings employ *section labels* and *expansion-lists* in order to identify how various passages are repeated and ordered.

When extracting a single strophe, either the abbreviated or through-composed versions can be used as input. However, when using the **strophe** command to select more than one strophe for output, it is essential that the input first be expanded to a through-composed version, via the **thru** command. For example, in order to select the first and third verses in the above passage by Schubert, the user would need to execute the following command pipeline:

```
thru wandern | strophe -s 1,3
```

The list following the **-s** option can contain individual strophes separated by commas. For example, the following command extracts verses 1, 3 and 4 in succession:

```
thru wandern | strophe -s 1,3,4
```

Strophes may also be output in non-numeric order as in the following command invocation:

```
thru wandern | strophe -s 4,3,2,1
```

If the **-x** option is invoked, **strophe** outputs only a single strophe whose string *label* is specified as an option. Strophe names need not be numerical. E.g.

```
strophe -x ossia
```

If the **strophe** command is invoked without any option, then all strophes are expanded in the output in numerical order beginning with strophe 1. Missing numerical strophes (such as a missing strophe `S/3` in a four-strophe encoding) will cause an error to be generated and terminate the **strophe** command.

Note that the **strophe** command allows strophe numbers containing a single decimal point, such as strophe `*S/4.2`. Having extracted the verse `*S/1`, the **strophe** command will output verse `*S/1.1` in preference to `*S/2` — if the decimal strophe is present. This feature allows more than one strophic passage to be encoded within a single abbreviated format file. This feature might prove useful, for example, in a musical work that contains a brief refrain in the middle of each verse.

The various strophe-related tandem interpretations are summarized below:

*strophe	strophic passage initiator
*S-	strophic passage terminator
*S/*string*	strophe name label
*S/*n[.n]*	strophe number label
*S/fin	strophe end indicator

Types of Strophe Interpretations

Note that for each strophic passage, all strophe labels must appear on the same record. See EXAMPLES below.

OPTIONS

The *strophe* command provides the following options:

-h	displays a help screen summarizing the command syntax
-s *strophe_list*	output numbered strophes according to *strophe_list*
-x *strophe_label*	output only strophes labelled *strophe_label*

Options are specified in the command line.

EXAMPLES

The following example is concocted to illustrate the operation of the **strophe** command. Consider the following Humdrum input:

```
!! 'strophe' example.
**example       **bar
*>[A,V,V,Coda]  *>[A,V,V,Coda]
*>A             *>A
A               i
*>V             *>V
*               **foo
*               *strophe
*               *^
*               *S/1            *S/2
B               1               2
*               *S/fin          *S/fin
*               *v              *v
*               *S-
*               **bar
C               refrain
*               *strophe
*               *^
*               *S/1.1          *S/2.1
B               1               2
*               *S/fin          *S/fin
*               *v              *v
*               *S-
*>Coda          *>Coda
*               **foo
E               i
*-              *-
```

Since this file is in abbreviated format, we must first expand it to through-composed form using the **thru** command. The resulting output is:

```
!! 'strophe' example.
**example       **bar
*thru           *thru
*>A             *>A
A               i
*>V             *>V
*               **foo
*               *thru
*               *strophe
*               *^
*               *S/1            *S/2
B               1               2
*               *S/fin          *S/fin
```

```
*               *v              *v
*               *S-
*               **bar
*               *thru
C               refrain
*               *strophe
*               *^
*               *S/1.1          *S/2.1
B               1               2
*               *S/fin          *S/fin
*               *v              *v
*               *S-
*>V             *>V
*               **foo
*               *thru
*               *strophe
*               *^
*               *S/1            *S/2
B               1               2
*               *S/fin          *S/fin
*               *v              *v
*               *S-
*               **bar
*               *thru
C               refrain
*               *strophe
*               *^
*               *S/1.1          *S/2.1
B               1               2
*               *S/fin          *S/fin
*               *v              *v
*               *S-
*>Coda          *>Coda
*               **foo
*               *thru
E               i
*-              *-
```

The command:

```
strophe file
```

will produce the following output:

```
!! 'strophe' example.
**example   **bar
*thru       *thru
*>A         *>A
A           i
*>V         *>V
*           **foo
*           *thru
B           1
*           **bar
*           *thru
C           refrain
B           1
*>V         *>V
*           **foo
*           *thru
B           2
*           **bar
*           *thru
C           refrain
B           2
*>Coda      *>Coda
*           **foo
*           *thru
E           i
*-          *-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

extract (4), **thru** (4), ***strophe** (2), **yank** (4)

NAME

synco — measure degree of metric syncopation

SYNOPSIS

synco [**-e**] [*inputfile* ...] [> *outputfile.syn*]

DESCRIPTION

The **synco** command characterizes the degree of metric syncopation evident at successive moments in a passage. It outputs a single Humdrum spine (`**synco`) containing numerical values representing the instantaneous level of syncopation. The **synco** command requires at least two spines of input data — one of which must be `**metpos`. (The `**metpos` representation encodes the position in the metric hierarchy for each data record in the input.) The other input spine(s) must contain information that implicitly or explicitly encodes the occurrence of note onsets. Appropriate inputs to **synco** include the pitch-related representations listed below. For descriptions of the various input representations refer to Section 2 *(Representation Reference)* of this reference manual.

`**cbr`	critical band rate (in equivalent rectangular bandwidths)
`**cents`	hundredths of a semitone with respect to middle C=0
`**cocho`	cochlear coordinates (in millimeters)
`**deg`	key-related relative scale degree
`**degree`	key-related absolute scale degree
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**pc`	pitch class
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)
`**solfa`	tonic solfa syllables
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **synco.**

The resulting `**synco` spine contains numerical values, where zero represents no metric syncopation and successively increasing values represent increasing amounts of metric syncopation.

The **synco** command implements a definition of metric syncopation inspired by the work of Lee and Longuet-Higgins (see REFERENCES). In brief, metric syncopation may be defined as a moment where an expected metric stress is absent. More specifically, a metrically syncopated moment is defined as occurring when no note-onset happens at a moment whose metric position is more important than that of the most recent note onset. For example, where a note onset occurs on the second beat of a quadruple meter, and is not followed by a note onset on the third beat, the third beat is deemed metrically syncopated because it occupies a higher metric position than the previous onset.

The numerical output values generated by **synco** are calculated as the logarithm of the metric position of the previous onset minus the logarithm of the metric position of the current moment — where the current moment has no note onset, and coincides with a higher metric position than the previous onset. The use of the logarithmic difference weights the output values so that missing downbeats at the beginning of a measure produce a greater metric syncopation value than lesser beats or sub-beats in the measure. In addition, metric syncopation is greater when the difference in metric position between the previous onset and the current moment is greatest. (See EXAMPLES.)

If more than one musical part is given as input, **synco** responds to the *aggregate* rhythmic structure — as though all of the parts were amalgamated into a single rhythmic stream. By itself, a single musical part may evoke considerable metric syncopation, but in combination with other parts, metrically syncopated moments are typically fewer. In short, running **synco** on a multi-part score normally produces different results from running **synco** on each part individually.

synco monitors the input in order to determine the Humdrum *time-base* — if encoded. Specifically, **synco** checks to ensure that the time-base is not excessively short. It is possible to have time-base values that exceed the temporal resolving power of human listeners. For example, if an onset appears a thirty-second duration prior to an expected down-beat, listeners are apt to hear the displaced onset as occurring **on** the beat rather than being a very short syncopation. "Excessively short" is operationally defined as a time-base resolution shorter than a sixteenth note. In such cases, **synco** issues a warning, noting that the time-base may be too short.

Note that only one metrically syncopated moment can happen following a given note onset; subsequent syncopated moments require the intervention of another note onset. By way of example, a note occurring immediately prior to an absent major downbeat, will not also cause syncopated moments to arise for other beats within a measure containing only rests. In short, two metrically syncopated moments can't occur without some note onset intervening.

It is recommended that output files produced using the **synco** command should be given names with the distinguishing '.syn' extension.

OPTIONS

The **synco** command provides the following options:

-e echo the input in the output
-h displays a help screen summarizing the command syntax

Options are specified in the command line.

If the **-e** option is invoked, the output will echo all of the input spines along with the `**synco` output.

EXAMPLES

The following two examples illustrate the use of **synco.** In both examples, the left-most spines represent the input, and the right-most spine (labelled `**synco`) represents the corresponding output.

The first example shows the minimum input of a single `**pitch` input plus the metric position information (`**metpos`). The meter signature is 2/4 and the time-base is an eighth duration. (See the **timebase (4)** command.) Hence there are 4 data records per measure. The first beat in each measure is assigned the metric position "1"; the second beat is assigned the metric position "2"; and the second half of each beat is assigned the metric position "3". Zero values in the `**synco` spine indicate the absence of any syncopation. In measure 3, a single syncopated moment happens at beat 2. The output was produced using the simple command: `synco inputfile`.

```
!! Example #1
**pitch   **metpos   **synco
*M2/4     *M2/4      *
*tb8      *tb8       *tb8
=1        =1         =1
r         1          0
.         3          0
r         2          0
A4        3          0
=2        =2         =2
G4        1          0
.         3          0
B4        2          0
r         3          0
=3        =3         =3
C5        1          0
C5        3          0
.         2          0.41
B4        3          0
=4        =4         =4
*-        *-         *-
```

In the following example, two metrically syncopated moments are evident. Notice that the rhythmic information for the two `**kern` spines is amalgamated, and that the non-pitch spine (`**foo`) has no affect on the processing.

```
!! Example #2
**foo   **kern   **metpos   **kern   **synco
*       *M2/4    *M2/4      *M2/4    *
*       *tb8     *tb8       *tb8     *tb8
A       =1       =1         =1       =1
A       4r       1          4r       0
A       .        4          .        0
A       .        3          .        0
A       .        4          .        0
A       8r       2          8r       0
A       .        4          .        0
A       [8a      3          [8a      0
A       .        4          .        0
A       =2       =2         =2       =2
A       4a]      1          8a]      1.10
A       .        4          .        0
A       .        3          8a       0
A       .        4          .        0
A       8b       2          8r       0
A       .        4          .        0
A       8r       3          8b       0
A       .        4          .        0
A       =3       =3         =3       =3
A       8cc      1          8cc      0
A       .        4          .        0
A       4.cc     3          4cc      0
A       .        4          .        0
A       .        2          .        0.41
A       .        4          .        0
A       .        3          8b       0
A       .        4          .        0
A       =4       =4         =4       =4
*-      *-       *-         *-       *-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

metpos (4), ****synco** (2), **timebase** (4), **urrhythm** (4)

REFERENCES

Longuet-Higgins, H. C., & Lee, C. S. "The perception of musical rhythms," *Perception*, Vol. 11 (1982) pp. 115-128.

NAME

tacet — reset MIDI channels to ensure silence

SYNOPSIS

tacet [**-i** *hex*]

DESCRIPTION

The **tacet** command resets the MIDI output. It sends "all-notes-off" commands on MIDI channels 1-16. The **tacet** command is useful if a MIDI application has terminated abnormally — leaving one or more sounding notes ("ciphers") on a MIDI device.

OPTIONS

The **tacet** command provides the following options:

-h	displays a help screen summarizing the command syntax
-i *hex*	assign MIDI input/output address to *hex*

Options are specified in the command line.

The **-i** option is used to specify the input/output address of the MIDI card. The default address is '330.' The address is specified as a hexadecimal number.

PORTABILITY

DOS 2.0 and up.

SEE ALSO

midreset (4), **perform** (4), **smf** (4)

REFERENCES

Use of the Music Quest Inc. MIDI library functions is gratefully acknowledged.

NAME

thru — expand abbreviated format representation to through-composed format

SYNOPSIS

thru [-v *version*] [*inputfile* ...]

DESCRIPTION

The **thru** command expands *abbreviated format* Humdrum representations to *through-composed formats* in which input passages are rearranged and output according to some specified *expansion list.*

Musical scores frequently contain notational devices such as repeat signs and Da Capos which permit more succinct renderings of a given document. Humdrum *section labels* and *expansion-lists* provide parallel mechanisms for encoding *abbreviated format* files. The **thru** command is normally used to expand various repetition devices. However, depending on the input, one of several expansions (dubbed *versions*) may be possible. Hence, **thru** is also useful for selecting a particular edition, performance, or interpretation from a composite input.

The input to **thru** must contain one or more sections identified by section labels. A *section* is a set of contiguous records. A *section label* is a tandem interpretation that consists of a single asterisk, followed by a greater-than sign, followed by a keyword that labels the section, e.g.

```
*>Exposition
*>Trio
*>Refrain
*>2nd ending
*>Coda
```

The section label *keywords* may contain any sequence of the following ASCII characters: the upper- or lower-case letters A–Z, the numbers 0 to 9, the underscore (_), dash (-), period (.), plus sign (+), octothorpe (#), tilde (~), at-sign (@), or space. All other characters are forbidden. A section label interpretation may occur anywhere in a Humdrum input, however, if more than one spine is present in a passage, identical section labels must appear concurrently in all spines.

Sections begin with a section label and end when either another section label is encountered, all spines are assigned new exclusive interpretations, or all spines terminate. If there is more than one spine present in a passage, identical section labels must appear concurrently in all spines.

An *expansion-list* is a tandem interpretation that takes the form of a single asterisk followed by a greater-than sign, followed by an optional label, followed by a list of section-labels enclosed in square brackets and separated by commas. Examples are given below. The first and second expansion-lists identify two section-labels in their lists. The last three expansion-lists have been labelled 'long,' '1955' and 'Czerny_edition' respectively.

```
*>[section1,2nd ending]
*>long[Exposition,Exposition]
*>1955[Aria]
*>Czerny_edition[refrain]
```

The **thru** command outputs each section in the order specified in the expansion list. If more than one expansion list is present in a file, then the desired *version* is indicated on the command line via the **-v** option. (See EXAMPLES.)

OPTIONS

The **thru** command provides the following options:

-h	displays a help screen summarizing the command syntax
-v *version*	expand the encoding according to expansion-list label *version*

Options are specified in the command line.

EXAMPLES

The following examples illustrate the operation of the **thru** command. Consider the following simple file:

```
**example     **example
*>[A,B,A,C]   *>[A,B,A,C]
*>A           *>A
data-A        data-A
*>B           *>B
data-B        data-B
*>C           *>C
data-C        data-C
*-            *-
```

This example contains just three data records — each of which has been labelled with its own section label. The file contains a single unlabelled expansion list which indicates that 'A' section should be repeated between the 'B' and 'C' sections. The following command:

```
thru inputfile
```

would produce the following "through-composed" output:

```
**example   **example
*thru       *thru
*>A         *>A
data-A      data-A
*>B         *>B
data-B      data-B
*>A         *>A
data-A      data-A
*>C         *>C
data-C      data-C
*-          *-
```

Notice that the expansion-list record has been eliminated from the output. A `*thru` tandem interpretation is added to all output spines immediately following each instance of an exclusive interpretation in the input. If `*thru` tandem interpretations are already present in the input, they are discarded (thus, running a file through **thru** twice will not change the file in any way). Also notice that there are now two sections in the output sharing the same label (`*>A`). This duplication of section-labels is not permitted in abbreviated-format encodings.

Consider the following more complex example. Imagine a *Da Capo* work in which a conventional performance procedes as follows: a first section ('A') is performed twice, followed by first and second endings — labelled 'B' and 'C' respectively. A subsequent section ensues ('D'), followed by a return to the first section ('A'). This first section is played just once, immediately followed by a final coda section ('E').

Imagine also a hypothetical performance of this work in which Murray Perahia makes three changes: Perahia repeats the 'D' section, he repeats the 'A' section when returning to the Da Capo — re-using the first ending before continuing to the coda following the repetition. Finally, Perahia has improvised an introduction to the work. Both the conventional interpretation and the hypothetical Perahia interpretation can be represented in the same encoded file as follows.

```
**example
*>Perahia[X,A,B,A,C,D,D,A,B,A,E]
*>Conventional[A,B,A,C,D,A,E]
*>X
data-X
*>A
data-A
*>B
!! 1st ending
data-B
*>C
!! 2nd ending
data-C
*>D
data-D
*>E
!! Coda
data-E
*-
```

The hypothetical Perahia version can be recreated by invoking the command:

```
thru -v Perahia inputfile
```

Alternatively, the "conventional" interpretation of the Da Capo structure could be produced by the command:

```
thru -v Conventional inputfile
```

In each case, the **thru** command will expand the input file according to the designated label for the expansion-lists. Note that there is no limit to the number of expansion-lists that may appear in a Humdrum file.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

strophe (4), **yank** (4)

WARNINGS

Humdrum ouput is not guaranteed with the **thru** command. In order to assure Humdrum output, it is necessary to have the same number of active spines at each point where sections are joined together in the expanded output. In addition, the exclusive interpretations must match where sections are joined.

Note that if an expansion list appears after some initial data records, then **thru** causes the initial material to be output before expanding the document according to the expansion list.

No two expansion lists can be identified using the same version label.

If two sections contain the same section label in the abbreviated document, then subsequent sections having the same label are ignored when expanded by **thru.**

Inputs that contain different section labels or expansion-lists in concurrent spines are illegal and will produce an error.

NAME

timebase — reformat `**kern` or `**recip` score with constant time-base

SYNOPSIS

timebase t *n*[.] [**-x**] [*inputfile* ...] [> *outputfile.tb*]

DESCRIPTION

The **timebase** command is used to reformat `**kern` or `**recip` inputs so that output data records represent equivalent slices (elapsed duration) of time. The effect of the **timebase** command is best illustrated by an example. With a specified time-base of a sixteenth duration, the following input:

```
**kern   **kern   **kern   **kern
4g        8.r      8.cc     16ee
.         .        .        8ff
.         32b      16cc     16gg
.         32a      .        .
8f        8cc      8dd      8ff
*-        *-       *-       *-
```

would produce the following output:

```
**kern   **kern   **kern   **kern
*tb16    *tb16    *tb16    *tb16
4g        8.r      8.cc     16ee
.         .        .        8ff
.         .        .        .
.         32b      16cc     16gg
8f        8cc      8dd      8ff
.         .        .        .
*-        *-       *-       *-
```

Each output record represents a snap-shot of a sixteenth duration following the previous data record. Depending upon the choice of time-base, the resulting output is either expanded or contracted in length. Details finer than the specified time-base are lost; in the above example, notice that the second thirty-second note (pitch 'A' in the second spine) has disappeared from the file as the time-base is only a sixteenth duration.

The time-base is selected by assigning a `**recip` duration value to the **-t** option. Time-base durations may be dotted.

Comments and barlines are preserved in the output, however, acciaccatura records (grace notes) are discarded.

It is recommended that output files produced using the **timebase** command should be given names with the distinguishing '.tb' extension.

OPTIONS

The **timebase** command provides a number of options.

-h displays a help screen summarizing the command syntax
-t *n* set time-base where *n* represents a `**recip` duration
-x strip duration values from the input

Options are specified in the command line. e.g.

```
timebase -t 8. -x
```

will remove `**kern` or `**recip` duration encodings from the output; each output data record will represent an elapsed duration of a dotted eighth note.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****kern** (2), **metpos** (4)

WARNINGS

Depending upon the defined time-base, passages of prolonged syncopation may disappear from the output. The **timebase** command assumes the integrity of the duration structure of the input score. Corrupt duration structures in the input will produce unpredictable results.

NAME

tonh — translate pitch-related representations to German pitch notation

SYNOPSIS

tonh [**-tx**] [*inputfile* ...] [> *outputfile.tnh*]

DESCRIPTION

The **tonh** command transforms various pitch-related inputs to the corresponding German system for designating pitches (Tonhöhe). It outputs one or more Humdrum `**Tonh` spines. German pitch designations are similar to the common A-G designations used by English speakers. The letter 'H' signifies the English 'B', whereas the letter 'B' signifies English 'B-flat'. Sharps and flats are indicated via the suffixes "is" and "es" respectively — hence 'Cis' for 'C#' and 'Ges' for 'Gb'. Special exceptions include 'Heses' for B double-flat rather than 'Bes', 'As' and 'Es' rather than 'Aes' or 'Ees', and 'S' as an alias for 'Es' (E-flat).

The **tonh** command is able to translate any of the pitch-related representations listed below. In each case, a tuning standard of A4 equals 440 hertz is assumed. For descriptions of the various input representations (including `**Tonh`) refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **tonh** command should be given names with the distinguishing '.tnh' extension.

`**cents`	hundredths of a semitone with respect to middle C=0
`**degree`	key-related scale degree
`**freq`	fundamental frequency (in hertz)
`**fret`	fretted-instrument pitch tablature
`**kern`	core pitch/duration representation
`**MIDI`	Music Instrument Digital Interface tablature
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0 (e.g. 12 = C5)
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)

Input representations processed by **tonh.**

OPTIONS

The **tonh** command provides the following options:

-h displays a help screen summarizing the command syntax
-t suppresses printing of all but the first ~~not~~ of a group of tied `**kern` notes
-x suppresses printing of non-Tonh data

Options are specified in the command line.

The **-t** option ensures that only a single output value is given for tied `**kern` notes; the output coincides with the first note of the tie.

In default operation, **tonh** outputs non-pitch-related signifiers in addition to the Tonhöhe value. For example, the `**pitch` token "Gb5zzz" will result in the output "Ges5zzz" — that is, after translating Gb5 to Ges5, the "zzz" signifiers are retained in the output. For some applications, echoing non-pitch-related signifiers in the output is useful. However, in other situations, the result can prove confusing — especially, when the non-pitch-related signifiers are numbers. Consider the case of the `**kern` token "8aa#"; after translating 'aa#' to Ais5, the non-pitch-related signifier '8' will also be output, hence the value 8Ais5 — which may cause confusion. Commands such as **pitch** and **tonh** treat the first number encountered in an input token as the octave designation. So further processing of this token may lead to it's interpretation as A#8 — or even A#85 — rather than A#5.

The **-x** option is useful for eliminating non-pitch-related signifiers from the output. For most `**kern` inputs, the **-x** option is recommended.

EXAMPLES

The following example illustrates the use of **tonh.** The input contains six pitch-related spines — two of which (`**degree` and `**cocho`) cannot be processed by **tonh.** In addition, there are two non-pitch-related spines (`**embell` and `**metpos`).

```
!! 'tonh' example.
**kern	**pitch	**MIDI	**deg	**metpos	**cocho	**degree	**embell
*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4	*M2/4
*	*	*	*	*	*	*d:	*
*	*	*	*	*tb8	*	*	*
=1	=1	=1	=1	=1	=1	=1	=1
8ee-	G#4foo	/60/bar	1foo	1	r	1/4	ct
.	.	/-60/	.	.	.	.	.
8ff	A3	/62/	2	3	9.89	2/4	upt
.	.	/-62/	.	.	.	.	.
8dd-	Ab3	/70/	1	2	7.07	3+/4	ct
.	.	/-70/	.	.	.	.	.
8d-	C#4	/61/	6	3	7.135	7/3	sus
.	.	/-61/	.	.	.	.	.
=2	=2	=2	=2	=2	=2	=2	=2
[4a-	r	.	5	1	r	r	.
.	.	.	7	3	5.5	1/4	ct
4a-]	D4	/48/ /52/	1	2	8.11	6+/4	ct
.	.	/-48/	.	.	.	.	.
.	D4 F4	/-52/	2	3	7.33 6.4	3/4 5/4	ct
=3	=3	=3	=3	=3	=3	=3	=3
r	G4	.	r	1	r	3/4 1/5	.
==	==	==	==	==	==	==	==
*-	*-	*-	*-	*-	*-	*-	*-
```

Executing the command

```
tonh -tx input > output
```

produces the following result:

```
!! 'tonh' example.
**Tonh    **Tonh    **Tonh    **deg    **metpos    **cocho     **Tonh    **embell
*M2/4     *M2/4     *M2/4     *M2/4    *M2/4       *M2/4       *M2/4     *M2/4
*         *         *         *        *           *           *d:       *
*         *         *         *        *tb8        *           *         *
=1        =1        =1        =1       =1          =1          =1        =1
Es5       Gis4      C4        1foo     1           r           D4        ct
.         .         .         .        .           .           .         .
F5        A3        D4        2        3           9.89        E4        upt
.         .         .         .        .           .           .         .
Des5      As3       B4        1        2           7.07        Fis4      ct
.         .         .         .        .           .           .         .
Des4      Cis4      Des4      6        3           7.135       Cis3      sus
.         .         .         .        .           .           .         .
=2        =2        =2        =2       =2          =2          =2        =2
As4       r         .         5        1           r           r         .
.         .         .         7        3           5.5         D4        ct
.         D4        C3 E3     1        2           8.11        H4        ct
.         .         .         .        .           .           .         .
.         D4 F4     .         2        3           7.33 6.4    F4 A4     ct
=3        =3        =3        =3       =3          =3          =3        =3
r         G4        .         r        1           r           F4 D5     .
==        ==        ==        ==       ==          ==          ==        ==
*-        *-        *-        *-       *-          *-          *-        *-
```

Both processed and unprocessed spines are output. Notice that the tied note at the beginning of measure 2 in the `**kern` spine has been rendered as a single note rather than as two notes (due to the **-t** option). Also notice that the non-pitch-related signifiers (e.g. foo) in the first notes of the `**pitch` and `**MIDI` spines have been stripped away (due to the **-x** option). In the case of the `**degree` input, **tonh** recognizes the spelling of various pitches in the context of the key of D minor. Hence, the raised third degree is Fis (F#), and the raised sixth degree is H (B natural).

FILES

The file `x_option.awk` is used by this program when the **-x** option is invoked.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****cents** (2), **cents** (4), ****degree** (2), **degree** (4), ****freq** (2), **freq** (4), ****fret** (2), **hint** (4), ****kern** (2), **kern** (4), ****MIDI** (2), **midi** (4), **mint** (4), ****pitch** (2), **pitch** (4), ****semits** (2), **semits** (4), ****solfg** (2), **solfg** (4), ****specC** (2) **specC** (4), ****Tonh** (2)

NAME

trans — transpose pitch representations

SYNOPSIS

trans -d [±]*n* [**-c** [±]*n*] [*inputfile* ...] [> *outputfile*]

DESCRIPTION

The **trans** command is used to transpose various Humdrum pitch representations. Transposed outputs maintain proper pitch-interval spellings, except in modal transposition; **trans** might be used to transpose a work up a perfect fourth, or down an augmented third, or from major mode to Phrygian. A "transposition interpretation" is appended to the output to mark the score as having been transposed.

The **trans** command is able to process those pitch-related representations listed below. All other inputs are simply echoed in the output. For descriptions of the various input representations refer to Section 2 *(Representation Reference)* of this reference manual.

`**kern`	core absolute pitch representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**solfg`	French solfège system (fixed 'doh')
`**Tonh`	German pitch system

Input representations processed by **trans**.

Transpositions are transformations that shift all pitch-signifier values up or down by some amount. Transpositions are specified by defining a *diatonic offset* and a *chromatic offset.* The diatonic offset affects only the pitch-letter name used to spell a note. The chromatic offset affects only the number of semitones shifted from the original pitch height.

For typical transpositions, both diatonic and chromatic offsets will need to be specified. For example, in transposing up a minor third (e.g. from C to E-flat), the diatonic offset is 'up two pitch-letter names,' and the chromatic offset is 'up three semitones.' The appropriate command invocation would be:

```
trans -d +2 -c +3 input > output
```

The plus signs above are optional; in their absence, **trans** assumes an upward transposition. Note that negative offsets can be mixed with positive offsets. For example, `trans -d -1 -c +1` will transpose the pitch C to B double-sharp, and F flat to E sharp, etc. (i.e. down one letter name, yet up one semitone).

Modal transpositions are invoked by simply omitting the chromatic offset information. (See

EXAMPLES below.)

It is recommended that output files produced using the **trans** command should be given names with the distinguishing '.tr' extension.

OPTIONS

The **trans** command provides the following options:

-c [±]*n*	transpose up(+) or down (-) *n* semitones
-d [±]*n*	transpose up(+) or down (-) *n* diatonic letter names
-h	displays a help screen summarizing the command syntax

Options are specified in the command line.

Note that the **-d** "option" is manditory rather than optional.

EXAMPLES

The following examples illustrate the use of **trans.**

Transposition up a minor third differs from transposition up an augmented second:

```
trans -d +2 -c +3 milhaud
trans -d +1 -c +3 milhaud
```

Enharmonic transpositions can be accomplished by defining the chromatic offset as zero semitones. For example, the following command transposes up a diminished second. It might be used to transpose from the key of C-sharp to the key of D-flat.

```
trans -d +1 -c 0 moonlight
```

Transposition up or down an octave requires both a diatonic offset and a corresponding chromatic offset of plus or minus 12 semitones. e.g.

```
trans -d -7 -c -12 guitar
```

In addition to exact pitch transpositions, **trans** can also perform *modal* transpositions. Modal transpositions arise when the pitch letter names are modified without regard for the precise semitone offset. To invoke a modal transposition, simply omit the chromatic offset specification. The following transposition changes diatonic pitch-letter names (down by two pitch-letters) so that the pitch C will become A, etc.

```
trans -d -2 major > aeolian
```

For inputs in major keys, the corresponding outputs will be in Aeolian mode.

For some applications, two or more successive transpositions may be necessary. For example, the following pipeline modifies inputs in major keys so that they are in the tonic

Dorian mode.

```
trans -d +1 major | trans -d -1 -c -2 > dorian
```

The first **trans** carries out a modal transposition — up the interval of a diatonic second. Thus, a work in D major would be placed in E Dorian. The second **trans** returns the score down the precise interval of a major second. Together, both transpositions would cause an input in F major to be transformed to F Dorian; an input in B-flat major would be transformed to B-flat Dorian, etc.

Whenever **trans** is invoked, it adds a tandem interpretation to the output indicating that the output representation has been transposed and is no longer at the original pitch. *Transposition tandem interpretations* are similar in syntax to the **trans** command itself. For example, the following tandem interpretation indicates that the score has been transposed up a major second.

```
*Trd1c2
```

The tandem interpretation in effect echoes the operation of the original transposition. If a score has undergone more than one transposition, the associated tandem interpretations will be ordered beginning with the most recent transposition. For example, a work that was transposed down a perfect fourth, followed by up a diminished second:

```
trans -d -3 -c -5
trans -d +1 -c 0
```

would contain the tandem interpretation:

```
*Trd1c0
*Trd-3c-5
```

SAMPLE OUTPUT

The following example illustrates the operation of **trans.** Given the following input:

```
**kern   **pitch  **Tonh   **solfg  **foo
=1       =1       =1       =1       .
c        C4       C4       do4      abc
c#       C#4      Cis4     do~d4    .
d-       Db4      Des4     re~b4    xyz
r        r        r        r        .
=2       =2       =2       =2       .
B-       Bb3      B3       si~b3    .
B--      Bbb3     Heses3   si~bb3   .
=3       =3       =3       =3       .
*-       *-       *-       *-       *-
```

the command

```
trans -d 1 -c 2
```

would produce the following output:

```
**kern    **pitch   **Tonh    **solfg   **foo
*Trd1c2   *Trd1c2   *Trd1c2   *Trd1c2   *
=1        =1        =1        =1        .
d         D4        D4        re4       abc
d#        D#4       Dis4      re~d4     .
e-        Eb4       Es4       mi~b4     xyz
r         r         r         r         .
=2        =2        =2        =2        .
c         C4        C4        do4       .
c-        Cb4       Ces4      do~b4     .
=3        =3        =3        =3        .
*-        *-        *-        *-        *-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

humsed (4), ****kern** (2), **kern** (4), ****pitch** (2), **pitch** (4), **recode** (4), ****solfg** (2), **solfg** (4), ****Tonh** (2), **tonh** (4)

WARNINGS

It is important not to confused transposed scores with notations of music for transposing instruments (such as the horn in F or the clarinet in B-flat). The `**pitch`, `**kern`, `**Tonh`, and `**solfg` representations are intended to represent absolute or concert pitch; transposed scores are deviate from this convention. The *transpose tandem interpretation* should not be used to indicate that an encoding is for a transposing instrument. A special tandem interpretation — beginning `*ITr` — is reserved for such designations. The interpretation `*Tr` means that the encoding no longer represents absolute or concert pitch. By contrast, the interpretation `*ITr` means that the instrumentalist reads from a score whose pitches are notated differently from concert pitch; nevertheless, the ensuing data is encoded at concert pitch. For example, a trumpet in B-flat plays a B-flat by fingering the pitch C. The absolute pitch (B-flat) is the proper `**kern`, `**pitch`, `**solfg`, or `**Tonh` encoding. Since the instrumentalist's *notation* is transposed up 1 diatonic letter-name and 2 chromatic semitones from the absolute or concert pitch, the encoded score will contain the instrument's transposition tandem interpretation

```
*ITrd1c2
```

(even though the encoded data is at concert pitch). Note that it is possible subsequently to transpose such a score using the **trans** command.

Note also that key and key signature tandem interpretations are not modified by **trans** since `**pitch`, `**kern`, `**solfg`, and `**Tonh`, intended to encode the original key and key signature at absolute pitch.

NAME

urrhythm — characterize Johnson-Laird rhythmic prototypes in a passage

SYNOPSIS

urrhythm *inputfile* [> *outputfile.urr*]

DESCRIPTION

The **urrhythm** command outputs a single Humdrum spine (`**URrhythm`) containing data representing rhythmic 'prototypes' (Ur-rhythms) evident in a musical passage. The command implements a variation of Johnson-Laird's theory of rhythmic prototypes (see REFERENCES below). In order to identify the rhythmic prototypes, **urrhythm** requires information about note onsets and metric position. These may be provided via two input spines: `**kern` (or `**recip`) and `**metpos`.

Johnson-Laird's rhythmic-prototype theory can be applied only to musical passages conforming to some established metric context, such as 2/4, 3/2, or 12/8 meters. The **urrhythm** command handles all regular types of meters (simple and compound, duple, triple and quadruple). Specifically, any meter having a "numerator" of 2, 3, 4, 6, 9, or 12 can be processed. **Urrhym** adapts to changes of meter, but is unable to handle irregular meters. If an irregular meter is encountered in the input an error is generated and the command terminates.

urrhythm characterizes each beat in a passage as belonging to one of three beat types: *Note* (N), *Syncopation* (S), or *Other* (O). Only major beats are characterized in this way. Hence, in 3/4 or 9/8 meters, three beats will be characterized for each complete measure. Similarly, in 4/2 and 12/16, four beats will be characterized for each complete measure.

A "Note" (signified in the output by the letter 'N') is defined as a beat that coincides with a note onset.

A "Syncopation" (signified by the letter 'S') is defined as arising when no note-onset happens on a beat whose position in the metric hierarchy is greater than that of the most recent note onset. By way of example, imagine a measure in 4/4 meter containing a quarter-note, followed by a half-note, followed by a quarter-note. The third beat position does not coincide with a note onset. The most recent note onset prior to the third beat occurs on beat two. Since beat three is a more important metric position than beat two, beat three is deemed to be syncopated.

Syncopated beats can happen only after the first note onset; subsequent syncopated moments will require another note onset (i.e. two syncopated moments can't occur in a row without some note onset intervening).

An "Other" (signified by the letter 'O') is any beat that is not a Note (N) or a syncopation (S).

It is recommended that output files produced using the **urrhythm** command should be given names with the distinguishing '.urr' extension.

OPTIONS

The **urrhythm** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

EXAMPLES

The following example illustrates the operation of **urrhythm.** The first two spines (`**kern` and `**metpos`) constitute the input. The third spine (`**URrhythm`) is added by the **urrhythm** command. All three spines are given in the output.

```
**kern   **metpos   **URrhythm
*M4/4    *M4/4      *M4/4
*tb8     *tb8       *tb8
8g       3          N
8g#      4          .
8a       2          N
[8cc     4          .
8cc]     3          S
8a       4          .
=1       =1         =1
4.cc     1          N
.        4          .
.        3          O
[8b-     4          .
4.b-]    2          S
.        4          .
.        3          O
[8g      4          .
=2       =2         =2
8g]      1          S
[8e-     4          .
8e-]     3          S
4.r      4          .
.        2          O
.        4          .
```

```
[4c     3       N
.       4       .
=3      =3      =3
2c]     1       S
.       4       .
.       3       O
.       4       .
4r      2       O
.       4       .
*-      *-      *-
```

WARNINGS

The **urrhythm** command is currently unable to handle Humdrum spine-path changes — such as join-path, exchange-path, or split-path. If spine-path changes are encountered an error is issued and the command terminates.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

****kern** (2), ****metpos** (2), **metpos** (4), ****recip** (2), **timebase** (4), **synco** (4), ****URrhythm** (2)

NOTE

The **urrhythm** command differs from Johnson-Laird's theory in the definition of syncopation. Johnson-Laird's theory requires that a listener be able to identify a syncopation retrospectively. That is, a listener is able to determine whether the current beat is a syncopation, only by determining what happens at the beginning of the next beat. The algorithm used here avoids the theoretical assumption of backward listening. (See Simpson & Huron, 1993.)

REFERENCES

Johnson-Laird, P. "Rhythm and meter: A theory at the computational level," *Psychomusicology*, Vol. 10 (1991) pp. 88-106.

Simpson, J. & Huron, D. "The perception of rhythmic similarity: A test of a modified version of Johnson-Laird's theory," *Canadian Acoustics*, Vol. 21, No. 3 (1993) pp. 89-90.

NAME

veritas — validate that a Humdrum document has not been modified

SYNOPSIS

veritas [**-v**] *inputfile*

DESCRIPTION

The **veritas** command provides an on-line means for formally or informally verifying that a given Humdrum file originates with a given publisher or source, or whether the file has been modified in some way. The command provides a convenient way of reassuring scholars of the accuracy or origin of a document.

The **veritas** command looks for a checksum validation number encoded in a VTS reference record in the given input file. (See the **Reference Records (1)** description.) The command then calculates the checksum for the file (excluding the VTS record itself) and compares this value with the checksum encoded in the file. If these values differ, a warning is issued that the file has been modified in some way. If these values are the same, a confirming message is issued.

Note that this verification process is easily circumvented by malicious individuals. For better security, the **-v** option should be invoked and the output checksum value should be compared with an independent printed list of checksums provided by the supplier of the electronic document.

OPTIONS

The **veritas** command provides the following options:

-h displays a help screen summarizing the command syntax
-v invoke verbose mode

Options are specified in the command line.

The **-v** option will cause the checksum validation number to be output. This value should be manually compared with a printed checksum provided by the supplier of the electronic document. Note that the change of even a single character in a file typically leads to a radically different checksum.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and POSIX.2 compliant **cksum** command.

SEE ALSO

cksum (UNIX), **sum** (UNIX)

NAME

vox — determine number of simultaneously active pitches

SYNOPSIS

vox [*inputfile* ...] [> *outputfile.vox*]

DESCRIPTION

The **vox** command calculates the number of tones sounding together at successive moments in time. It outputs a single Humdrum spine (`**vox#`) where successive integers indicate the total number of concurrently sounding pitches for each data record. Multiple-stops are properly supported.

The **vox** command accepts as input any pitch-encoded Humdrum representations listed below. For descriptions of the various input representations refer to Section 2 *(Representation Reference)* of this reference manual.

It is recommended that output files produced using the **vox** command should be given names with the distinguishing '.vox' extension.

`**cbr`	critical band rate (in equivalent rectangular bandwidths)
`**cents`	hundredths of a semitone with respect to middle C=0 (e.g. 1200 equals C5)
`**cocho`	cochlear coordinates (in millimeters)
`**deg`	key-related relative scale degree
`**degree`	key-related absolute scale degree
`**freq`	fundamental frequency (in hertz)
`**kern`	core pitch/duration representation
`**pc`	pitch class representation
`**pitch`	American National Standards Institute pitch notation (e.g. "A#4")
`**semits`	equal-tempered semitones with respect to middle C=0
`**solfa`	tonic solfa syllables
`**solfg`	French solfège system (fixed 'doh')
`**specC`	spectral centroid (in hertz)
`**Tonh`	German pitch system

Input representations processed by **semits.**

OPTIONS

The **vox** command provides only a help option:

-h displays a help screen summarizing the command syntax

Options are specified in the command line.

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

vox# (2)

NAME

xdelta — calculate sequential numeric differences between successive data tokens

SYNOPSIS

xdelta [**-ae**] [**-b** *regexp*] [**-s** *regexp*] [*inputfile* ...]

DESCRIPTION

The **xdelta** command calculates the numeric differences between successive data tokens within individual spines. By way of illustration, **xdelta** will change a sequence of numerical tokens — such as *x, y, z* — to their successive differences *y–x, z–y* (i.e. Δx). The **xdelta** command might be used for such purposes as determining the melodic interval distances between successive pitches in a musical line, or calculating changes of duration for successive notes.

Each output interpretation is automatically assigned a new name by prepending the letter 'X' to the input interpretations. For example, an input of `**semits` will result in an output interpretation named `**Xsemits`. Output from **xdelta** may be reprocessed as input, so that the 'differences between the differences' (i.e. second derivative) may be calculated. Once again, the letter 'X' will be prepended to the input interpretation, so that an input interpretation such as `**Xfreq` will result in an output interpretation named `**XXfreq`.

The **xdelta** command has a number of subtleties in its operation that facilitate the processing of music-related data. However, these subtleties can lead to unexpected results for the inexperienced user; caution is advised.

Numeric data are processed by calculating the arithmetic difference between successive data tokens within each spine. Hence, an input token '`-5`' followed by '`-2`' will result in an output of '`3`'. Positive output values indicate increasing input values; negative output values indicate decreasing input values. Null tokens are simply echoed in the output, however numerical processing continues as if the null tokens were absent. For example, the input token '`-5`' followed by a null token, followed by '`-2`' will result in an output of '`3`'.

No difference value is calculated for the first numeric input token; however, the numeric value of the first numeric token is echoed in the output — appearing in square brackets. These initial values are referred to as *offset* values, since they indicate the starting value from which subsequent differences are calculated. Offset values can prove useful in attempting to reconstruct the input, but the user may wish to eliminate offset values in subsequent processing (see below).

All data tokens containing only non-numeric data are simply echoed in the output. In the default operation, data tokens without numbers cause the difference calculation to be suspended, and the next occurrence of a numeric token is treated as a new offset value for subsequent calculations. As noted, null-tokens (.) are also directly echoed. If data tokens contain both numeric and non-numeric data (e.g. `42abc`), the default operation is to suppress the echoing of non-numeric signifiers in the corresponding output. Hence, an input token '`33b`' followed by '`p51.3xm`' will result in the output token '`18.3`' (i.e. 51.3 – 33). However, the accompanying non-numeric data can be echoed in the output by invoking the **-e** option.

The **xdelta** command is able to calculate unsigned (absolute) values where appropriate — using the **-a** option. It is also able to handle multiple stops and data-flow interruptions such as the occurrence of barlines. By defining *regular expression* patterns, the user may select which types of data tokens should be ignored by **xdelta.** (See EXAMPLES below.)

Note that the output spine generated by **xdelta** preserves the same record-type structure as the input, and so may readily be pasted with the input file using the Humdrum **assemble** command.

OPTIONS

The **xdelta** command provides the following options:

-a	output absolute difference values
-b *regexp*	break; do not calculate difference for tokens matching *regexp;* restart difference calculations with next numerical token
-e	echo non-numeric data for input tokens containing numbers
-h	displays a help screen summarizing the command syntax
-s *regexp*	skip; completely ignore tokens matching *regexp;* (echo in output only)

Options are specified in the command line.

The "skip" function takes precedence over the "break" function, so input strings matching both the skip (-s) and break (-b) regular expressions cause a skip rather than a break.

EXAMPLES

The various aspects of the **xdelta** command are best illustrated using a set of examples. Consider the following input:

```
**cents
100
300
.
1200
600
r
-200
1000
*-
```

Using the default invocation, the **xdelta** command transforms the above input into the following output:

```
**Xcents
[100]
200
.
900
-600
r
[-200]
1200
*-
```

Notice that the input interpretation (`**cents`) has been modified to `**Xcents` in the first record. As can be seen, the leading or "offset" value '`100`' has been echoed in the second record — although it has been printed in square brackets. This is not a "difference" value since there is no previous numerical value from which to calculate a difference; **xdelta** simply echoes the initial starting value. The third output record contains the value '`200`' — which is the difference between the second and third input records (300 minus 100). (Musically, we would say that the difference between 100 cents above middle C followed by 300 cents above middle C is an increase of plus 200 cents.) The null-token in the fourth record has been echoed. Null-tokens have no effect on subsequent numerical calculations and are treated as though they are non-existent. Thus the fifth output record contains the difference between the third and fifth input records (1200 minus 300 equals 900). The sixth input record ('`600`') is lower in value than the preceding value ('`1200`') and so produces a negative output (600 minus 1200 equals –600). The seventh input record contains no numerical value; as a result, **xdelta** "breaks" operation; it cannot calculate a numerical difference value. The output action is to echo the input token ('`r`') and to begin looking for a new offset value. The eighth input record ('`-200`') begins a new sequence of numerical values; the output echoes `[-200]` as the new offset. The ninth input record (`1000`) is 1200 cents above –200, and so the corresponding output value is 1200.

Sometimes numerical values appear in tokens that the user doesn't want processed. A good example occurs with numbered barlines. Consider the following simple example.

```
**dur
1.6
=1
2.5
*-
```

If the **xdelta** command is invoked with the default options, the output will be:

```
**Xdur
[1.6]
-0.6
1.5
*-
```

In other words, the measure number (1) interacts (incorrectly) with the duration values. This can be avoided by using the **-s** (skip) option. The skip option allows the user to identify records that should not be involved in Δx processing. The `**dur` barline signifier is the equals-sign; hence, the command `xdelta -s = input` will cause the barlines to be ignored in the numerical calculation, and so produce the following (correct) output:

```
**Xdur
[1.6]
=1
0.9
*-
```

Some inputs may contain multiple-stops — that is, Humdrum data tokens containing two or more encoded components. The **xdelta** command is also able to process numerical data tokens containing multiple-components. Consider, for example, the following `**semits` file:

```
**semits
3
4 7
-3 -7 11
12
*-
```

Notice the presence of the double- and triple-stops in the fourth and fifth records. Using the default invocation, the **xdelta** command processes this input as follows:

```
**Xsemits
[3]
1 4
-7 (-11) (-14) 4
15 19 1
*-
```

Once again, the input interpretation (`**semits`) has been modified to `**Xsemits`. The leading or offset value [3] has been echoed in the second record. (The user might wish to eliminate such offset values via the **humsed** command; see below.) The third records in both the input and output contain double-stops. In the output, the first value of the double-stop

('1') represents the difference between 3 and 4. The second value in the double-stop ('4') represents the difference between 3 and 7. In short, **xdelta** traces both possible difference "paths." In moving from the pitch D# to two concurrent pitches (E and G), we may trace both the D#-E interval (1 semit) and the D#-G interval (3 semits).

In processing successive multiple-stops **xdelta** does not calculate all of the possible permutations. For example, in the case of two consecutive triple-stops, **xdelta** will calculate three intervals corresponding to the first notes in both triple-stops, the second notes, and the third notes.

Where the number of multiple-stops changes, **xdelta** operates under some special conventions. Consider, for example, the case of a double-stop followed by a triple-stop: the pitches P+Q followed by X+Y+Z. All of the possible (interval) differences might be enumerated as follows: X-P, Y-P, Z-P, X-Q, Y-Q and Z-Q. **Xdelta** first calculates the "outer" interval distances (X–P and Z–Q). It then calculates a permuted set of "inner" intervals (Y–P and Y–Q). The remaining intervals are considered unlikely or implausible and are not calculated by **xdelta.**

In the above example, moving from the double-stop to the triple stop between records three and four generates the two "outer" interval distances (-3 minus 4 equals -7; 11 minus 7 equals 4), as well as the permuted "inner" intervals (-7 minus 4 equals -11; -7 minus 7 equals -14). Both the resulting inner intervals are printed in parentheses. A similar process occurs when moving from records four to five. Three intervals may be traced from the 3 initial tokens to the subsequent single token.

Depending on the goal, the presence of the parentheses makes it easy for the user to eliminate the inner intervals using the Humdrum stream-editor **humsed.** For example, the command:

```
humsed 's/([^)]*) //g' input > output
```

can be used to eliminate inner intervals. Alternatively, the command:

```
humsed 's/[()]//g' input > output
```

can be used to eliminate the parentheses surrounding the inner intervals. Offset values can be transformed to null-tokens using the command:

```
humsed 's/\[[^]]*\]/./g' input > output
```

Entire records containing offset values can be eliminated using the command:

```
humsed '/\[.*\]/d' input > output
```

A further example shows how the output of **xdelta** can be recirculated as input, and the second derivative calculated. In the example below, the first spine is the original input, consisting of a rising-falling major arpeggio, followed by an ascending major scale. The second spine is the corresponding output from the command:

```
xdelta -s = spine1 | humsed 's/\[.*\]/./' > spine2
```

The original input and both outputs have been assembled together below.

Notice that barlines have been skipped and that the initial offset value has been changed to a null token (using **humsed**). The second spine has then been used as input to **xdelta** with the result of the following command given in the third spine.

```
xdelta -a -s = spine2 | humsed 's/\[.*\]/./' > spine3
```

Notice that only absolute numerical differences have been generated in the third spine. In the output below, semitone pitch values are coordinated with the interval by which it was approached (2nd spine) and by the change of interval size (3rd spine). Notice that large values in third spine (e.g. 10 and 6) correspond to points in the input where the arpeggio changes direction, and where the arpeggio changes to a scale. (It is common to encounter such large discontinuities whenever a pattern changes.)

```
**semits   **Xsemits   **XXsemits
*M3/4      *M3/4       *M3/4
=1         =1          =1
0          .           .
4          4           .
7          3           1
=2         =2          =2
12         5           2
7          -5          10
4          -3          2
=3         =3          =3
0          -4          1
2          2           6
4          2           0
5          1           1
=4         =4          =4
7          2           1
9          2           0
11         2           0
=5         =5          =5
12         1           1
====       ====        ====
*-         *-          *-
```

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

assemble (4), **humsed** (4), **mint** (4), **recode** (4), **regexp** (4), **regexp** (6), **ydelta** (4)

NAME

yank — extract passages from a Humdrum input

SYNOPSIS

yank [**-c**] **-l** **-r** *range* [*inputfile* ...]
yank [**-c**] **-m** *regexp* **-r** *range* [*inputfile* ...]
yank [**-c**] **-n** *regexp* **-r** *range* [*inputfile* ...]
yank [**-c**] **-o** *regexp* [**-e** *regexp*] **-r** *range* [*inputfile* ...]
yank [**-c**] **-s** *section_label* **-r** *range* [*inputfile* ...]

DESCRIPTION

The **yank** command permits the selective extraction of segments or passages from a Humdrum input. Yanked material may consist of specified input records — addressed by absolute line number, or relative to some marker. In addition, **yank** is able to output logical segments (such as measures, phrases, or labelled sections), and is able to output material according to content. The output always consists of complete records; **yank** never outputs partial contents of a given input record.

OPTIONS

The **yank** command provides the following options:

-c	include all comments prior to the yanked material in the output
-e *regexp*	define end-delimiter for yanked segments as *regexp*; used with **-o**
-h	displays a help screen summarizing the command syntax
-l	yank all lines whose line numbers appear in **-r** *range*
-m *regexp*	yank lines matching *regexp* listed in **-r** *range*
-n *regexp*	yank segments delineated by *regexp* according to cardinal **-r** *range*
-o *regexp*	yank segments delineated by *regexp* according to ordinal **-r** *range*
-r *range*	yank section in ranges listed in *range*; used with **-l**, **-m**, **-n**, **-o** and **-s**
-s *section*	yank section labelled *section* according to **-r** *range*

Options are specified in the command line.

The simplest operation for **yank** occurs when the **-l** option is specified. In this case **yank** echoes those lines in the input stream whose line-numbers appear in a specified *range*. The *range* consists of one of more integers separated by commas; inclusive ranges can be specified by separating two integers by a dash (-). For example, the following command selects lines, 5, 13, 23, 24, 25, and 26 from the file named `dvorak`:

```
yank -l -r 5,13,23-26 dvorak
```

The dollar sign ($) can be used to refer to the last record in the input. For example, the following command yanks the first and last records from the file `verdi`.

```
yank -l -r '1,$' verdi
```

(Note that single quotes may be needed in regular expressions and range specifications in order to prevent the shell from misinterpreting characters such as the dollar sign or the asterisk.) Records prior to the end of the input can be specified by subtracting some value from $. For example, the following command yanks the first 20 records of the last 30 records contained in the file `ginastera`. (Notice that the dash/minus sign is used both to convey a range and as an arithmetic operator.)

```
yank -l -r '$-30-$-10' ginastera
```

In addition to the specified output lines, **yank** also outputs interpretations and comments as described below (see INTERPRETATIONS AND COMMENTS).

Alternatively, **yank** can output lines relative to some user-defined *marker*. This mode of operation can be invoked using the **-m** option. Markers are specified as regular expressions. The following command outputs the first and third data records following each occurrence of the string "XXX" in the file `wieck`.

```
yank -m XXX -r 1,3 wieck
```

The **-r** option is used to specify the range. If the value zero ("0") is specified in the range, then the record containing the marker is itself output.

Since markers are interpreted by **yank** as regular expressions, complex markers can be defined. For example, the following command yanks the first data record following all occurrences of any record in the file `franck` beginning with a letter and ending with a number:

```
yank -m '^[a-zA-Z].*[0-9]$' -r 1 franck
```

In musical applications, it is often convenient to yank material according to logical segments such as measures or phrases. In order to access such segments, the user must specify a segment *delimiter* using the **-o** or the **-o** and **-e** options. For example, common system barlines are represented by the presence of an equals-sign (=) at the beginning of a data token. Thus the user might yank specific measures from a `**kern` file by defining the appropriate barline delimiter and providing a range of (measure) numbers. Consider the following command:

```
yank -o ^= -r 1,12-13,25 joplin
```

Unlike the **-m** option, the **-o** option interprets the range list as ordinal occurrences of segments delineated by the delimiter. Whole segments are output rather than specified records as is the case with **-m.** As in the case of markers, delimiters are interpreted according to regular expression syntax. Each input record containing the delimiter is regarded as the *start* of the next logical segment. In the above command, the command-line range specifies that the first, twelfth, thirteenth, and twenty-fifth logical segments (measures) are to be yanked from the file named `joplin`. All records starting with the delimiter record are output up to, but not including the next occurrence of a delimiter record.

Where the input stream contains data prior to the first delimiter record, this data may be addressed as logical segment "zero." For example,

```
yank -o '^=' -r 0 mahler
```

can be used to yank all records prior to the first common system barline. With the **-o** option, notice that *actual* measure numbers are irrelevant: **yank** selects segments according to their *ordinal* position in the input stream rather than according to their *cardinal* label.

When the **-n** option is invoked, however, **yank** expects a numerical value to be present in the input immediately following the user-specified delimiter. In this case, **yank** selects segments based on their numbered label rather than their ordinal position in the input. For example,

```
yank -n ^= -r 12 goldberg
```

will yank all segments begining with the label `=12` in the input file `goldberg`. If more than one segment carries the specified segment number(s), all such segment are output. Note that the dollar-sign anchor cannot be used in the range expression for the **-n** option. Note also that input tokens containing non-numeric characters appended to the number will have no effect on the pattern match. For example, input tokens such as `=12a,` `=12b`, or `=12;` will be treated as equivalent to `=12`.

As in the case of the **-o** option, the value zero ("0") addresses material prior to the first delimiter record. Like the **-o** option, the value zero may be reused for each specified input file. Thus, if `file1,` `file2` and `file3` are Humdrum files:

```
yank -n ^= -r 0 file1 file2 file3
```

will yank any leading (anacrusis) material in each of the three files.

When the -s option is invoked, **yank** extracts passages according to Humdrum section labels encoded in the input. Humdrum section labels are tandem interpretations that conform to the syntax:

*>*label_name*

Labels are frequently used to indicate formal divisions, such as, coda, exposition, bridge, second ending, trio, minuet, etc. The following command yanks the second instance of a section labelled `First Theme` in the file `haydn08`:

```
yank -s 'First Theme' -r 2 haydn08
```

Note that with "through-composed" Humdrum files it is possible to have more than one section containing the same section-label. (See the **thru** command.)

INTERPRETATIONS AND COMMENTS

If **yank** is given a Humdrum input, it always produces a syntactically correct Humdrum output. All interpretations prior to and within the yanked material are echoed in the output. The **yank** command also appends the appropriate spine-path terminators at the end of the yanked segment.

Any comments *prior* to the yanked passage may be included in the output by specifying the **-c** option.

EXAMPLES

The following examples illustrate how the **yank** command may be used.

```
yank -l -r 1120 messiaen
```

yanks line 1120 in the file `messiaen`.

```
yank -n ^= -r 27 sinfonia
```

yanks numbered measures 27 from the `**kern` file `sinfonia`.

```
yank -n ^= -r 10-20 minuet waltz
```

yanks numbered measures 10 to 20 from *both* the `**kern` files `minuet` and `waltz`.

```
yank -o ^= -r '0,$' fugue ricercar
```

yanks any initial anacrusis material plus the final measure of both `fugue` and `ricercar`.

```
cat fugue ricercar | yank -o ^= -r '0,$'
```

yanks any initial anacrusis material from the file `fugue` followed by the final measure of `ricercar`.

```
yank -n 'Rehearsal Marking ' -r 5-7 fugue ricercar
```

yanks segments beginning with the strings `"Rehearsal Marking 5,"` `"Rehearsal Marking 6,"` and `"Rehearsal Marking 7."` Segments are deemed to end when a record is encountered containing the string `"Rehearsal Marking "`.

```
yank -o { -e } -r '1-$' webern
```

yanks all segments in the file `webern` beginning with a record containing "{" and ending with a record containing "}." The command:

```
yank -o { -e } -r '1-4,$-3-$' faure
```

yanks the first four and last four segments in the file `faure` — where segments begin with an open brace ({) and end with a closed brace (}). In the `**kern` representation, this would extract the first and last four phrases in the file.

```
yank -s Coda -r 1 stamitz
```

will yank the first occurrence of a section labelled `Coda` in the file `stamitz`.

WARNINGS

Where integer ranges are specified in the **yank** range-list, overlapping values are collapsed. For example, the command `yank -l -r 5-7,6-8` is interpreted as equivalent to `yank -l -r 5-8`; lines 6 and 7 will be echoed only once in the output stream. If the user wishes to have multiple occurrences of material in the output stream, the **yank** command can be invoked more than once (e.g. `yank -l -r 5-7 ...; yank -l -r 6-8 ...`).

Note that yanked segments cannot be output in an order other than their order in the input. For example, assuming that measure numbers in an input stream increase sequentially, **yank** is unable to output measure number 6 prior to outputting measure number 5. Once again, the order of output material can be rearranged by invoked the **yank** command more than once (e.g. `yank -l -r 100 ...; yank -l -r 99 ...; yank -l -r 98 ...`).

In the case of the **-m** option, note that range elements cannot address records more than one marker away from the current marker. For example, in a file where markers occur every 10 records, range expressions such as `'25'` and `'$-17'` will result in no output. In addition, range expressions such as `'1-25'` and `'$-17-$'` will have the same effect as `'1-10'` and `'$-9-$`. Note also that for the same input file, the range expression `'6-$-7'` will result in no output.

FILES

The files `find_reg.awk`, `findpair.awk` and `number.awk` are used by **yank.**

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

awk (UNIX), **extract** (4), **grep** (UNIX), **egrep** (UNIX), **patt** (4), **pattern** (4), **regexp** (4), **regexp** (6), **timebase** (4), **thru** (4)

NAME

ydelta — calculate numeric differences for concurrent data

SYNOPSIS

ydelta [-s *regexp*] **-i** '***interpretation*' [*inputfile*]

DESCRIPTION

The **ydelta** command outputs a single spine containing numeric data representing the numeric differences for concurrent tokens in specified input spines. For example, **ydelta** may be used to characterize the semitone distances separating pitches in a given vertical sonority or chord.

The user indicates which spines in the input stream are to be processed by specifying an interpretation via the **-i** option. The output interpretation is automatically assigned a new name by prepending the upper-case letter 'Y' to the given input interpretation. For example, a specified input of `**semits` will result in an output interpretation named `**Ysemits`.

OPTIONS

The **ydelta** command provides the following option:

-h	displays a help screen summarizing the command syntax
-i ***interp*	process input spines with interpretations ***interp*
-s *regexp*	skip over records matching *regexp;* If all target spines contain the same token; token is output. If the tokens differ, the entire record is output (although all tabs are replaced by spaces).

Options are specified in the command line.

EXAMPLES

The operation of **ydelta** is best illustrated using an example. Consider the following input file called `praetorius`:

```
!! Praetorius, Es ist ein' Ros' entsprungen.
**semits   **semits   **text   **semits   **kern
!bass      !tenor     !text    !alto      !soprano
*M2/2      *M2/2      *M2/2    *M2/2      *M2/2
*F:        *F:        *        *F:        *F:
```

```
-7      0       Es      9       1cc
-7      0       ist     9       2cc
-7      -3      ein'    5       2cc
-2      -2      Ros'    5       2dd
-7      -3      ent-    5       2cc
-12     -5      sprun-  4       1cc
-10     -7      gen     2       1a
-14     -7      aus     2       1b-
-7      -3      ei-     0       2a
-8      0       ner     .       1g
-10     (-2     Wur-    2       .
.       -3)     .       .       1f
.       .       .       -3      .
-12     .       zel     0       .
.       -5      .       .       2e
-19     -3      zart,   0       1f
=       =       =       =       =
*-      *-      *-      *-      *-
```

Given the following command line:

```
ydelta -i '**semits' praetorius > chords
```

the above input file would result in the following output:

```
!! Praetorius, Es ist ein' Ros' entsprungen.
**Ysemits
!
*
*
[-7] 7 16
[-7] 7 16
[-7] 4 12
[-2] 0 7
[-7] 4 12
[-12] 7 16
[-10] 3 12
[-14] 7 16
[-7] 4 7
[-8] 8
[-10] 8 12
[-3]
[-3]
[-12] 12
```

```
[-5]
[-19] 16 19
.
*-
```

Only those input spines labelled `**semits` have been processed; both `**text` and `**kern` have been ignored. The first value (given in square brackets) represents the lowest numerical value found in the processed spine(s) for the current data record. Successive numerical values indicate the differences between the lowest value and the remaining numeric values in the other spines. For example, the first input record contains the `**semits` values -7, 0, and 9. The lowest of these values is -7 — which is placed in square brackets. The next lowest value (0) is 7 units above the lowest value, whereas the value 9 is 16 units above the lowest value. These other values are sorted and printed in ascending order in the output token. Notice that the output is typically in the form of Humdrum multiple stops. If only one numerical value is present in the processed input, a single value (in square brackets) will be output. If no numerical values are present, a null token (.) is output (see for example, the barlines).

PORTABILITY

DOS 2.0 and up, with the MKS Toolkit. OS/2 with the MKS Toolkit. UNIX systems supporting the *Korn* shell or *Bourne* shell command interpreters, and revised *awk* (1985).

SEE ALSO

hint (4), **regexp** (4), **regexp** (6), **xdelta** (4)