UNIX Tools for Musical Research

The Humdrum Toolkit

◇ REFERENCE MANUAL ◇

David Huron

Center for Computer Assisted Research
in the Humanities
525 Middlefield Road, Ste 120
Menlo Park, CA 94025

UNIX Tools for Musical Research

The Humdrum Toolkit

◇ REFERENCE MANUAL ◇

David Huron

Center for Computer Assisted Research
in the Humanities
525 Middlefield Road, Ste 120
Menlo Park, CA 94025

Library of Congress Cataloguing in Publication Data

Huron, David Brian, 1954-
Unix Tools For Musical Research: The Humdrum Toolkit;
Reference Manual.

551pp.

Distributed by the *Center for Computer Assisted Research in the Humanities.*

Menlo Park, CA

Software Design: Prof. David Huron
Music Department
Conrad Grebel College
University of Waterloo
Waterloo, Ontario
Canada N2L 3G6

Software Queries: (519) 885-0220 ext. 247
E-mail: dhuron@watserv1.uwaterloo.ca

Software Distribution: Center for Computer Assisted Research in the Humanities
525 Middlefield Road,
Suite 120
Menlo Park, California
U.S.A. 94025

FTP access: archive.uwaterloo.ca
Telephone orders: (415) 322-3307
FAX orders: (415) 329-8365
E-mail orders: ccarh@netcom.com

Financial Assistance: Distribution of the Humdrum Toolkit has been supported in part by a grant from the Social Sciences and Humanities Research Council of Canada.

Humdrum Programmers: The Humdrum Toolkit was originally programmed by David Huron. It was expanded and re-programmed by Tim Racinsky and Kyle Dawkins. Additional programming was done by Keith Orpen and Keith Mashinter. This reference manual was written by David Huron.

Preface

As a music scholar, I have long appreciated the gains in personal productivity that are possible through the judicious use of computer technology. At the same time I've come to recognize that computers are inherently interesting objects, and that there is always the danger of finding the technology more interesting than the musical problems. While musicological problems are often elusive and enigmatic, software programming can offer the quick rewards of linear thinking and the illusion of boundless power. For this reason, I am wary of computer expertise *per se*. My hope is that this manual testifies, not to a compelling interest in computers, but to a more abiding interest in understanding music.

The mindless tasks that often dominate scholarly projects can be powerful deterents for those contemplating research. Seeing other scholars discouraged by such mindlessness has encouraged me to share my own research tools with the wider scholarly community. The distribution of this software is accompanied, however, by some trepidation. Any set of tools requires the development of a concomitant expertise, and the Humdrum Toolkit is no exception. It is my hope that the investment of time required to learn how to use Humdrum will be more than offset by the subsequent scholarly gains. Regrettably, not all Humdrum users may have this experience.

The Humdrum Toolkit remains a work in progress. As will become evident, the very structure of Humdrum invites users to intervene, augment, re-define, throw-away, and otherwise tinker. Users are encouraged to formulate their own representation schemes and to create their own task-specific software tools — as appropriate. In the case of predefined Humdrum representations, comments, criticisms, and suggests for improvement are most welcome.

A number of individuals have contributed directly or indirectly to the creation of the Humdrum Toolkit. I am deeply indebted to my supporters and collaborators, and take pleasure in voicing my public thanks.

Research assistants Tim Racinsky and Kyle Dawkins have been indispensable in bringing Release 1.0 to completion. Tim Racinsky revised nearly all of my original code and programmed innumerable new tools as well. I have especially valued Tim's cautious skepticism and his relentless pursuit of design inconsistencies and bugs. Kyle Dawkins wrote all of the MIDI-related code and also handled system administration. Through his musical savvy and high energy, Kyle contributed significantly to the excitement of the project.

In addition, several other individuals have contributed by volunteer efforts to this project. Dr. Gregory Sandell kindly agreed to include his library of spectral analyses of instrument tones in this release. Keith Orpen implemented the Damerau-Levenshtein metric used in the program for measuring non-parametric similarity. Keith Mashinter contributed the Kameoka and Kuriyagawa dissonance tool. Jasba Simpson co-authored the implementation of the modified Johnson-Laird syncopation algorithm. Andreas Kornstaedt helped with portability problems. Simon Clift programmed some exploratory routines that helped clarify the feasibility (and infeasibility) of certain tools.

Timothy Prime and Maki Ishizaki volunteered their time to test several of the tools and critique parts of the accompanying documentation. Sandra Serafini put the toolkit through its paces while pursuing her undergraduate thesis on Haydn string quartets. Stephen Bondy suggested useful modifications to the representation for fretted instruments.

Randall Howard, President of Mortice Kern Systems, provided valuable professional advice on software testing and project organization. In addition, Mortice Kern Systems donated critical development and documentation software that saved a great deal of time. I am also indebted to Dr. Walter Hewlett and Dr. Eleanor Selfridge-Field of the Center for Computer Assisted Research in the Humanities for their professional support and encouragement.

Finally, I would like to thank the Social Sciences and Humanities Research Council of Canada for providing funding to assist in the release of this software to the scholarly community.

David Huron
1994 August
Waterloo, Ontario

CONTENTS

Preface 1

Introduction *3*
- Organization of Reference Information *3*
- Commands and Representations *4*
- Documentation Style *4*

1. Humdrum General Introduction 7

Introduction *7*
The Humdrum Syntax *8*
Spine Paths *15*
The Humdrum Syntax: A Formal Definition *19*
Section Labels and Repetitions *22*
Versions *23*
Strophic Representations *24*
The Humdrum Syntax: Conclusion *26*
Reference Records *26*
- Authorship Information *28*
- Performance Information *30*
- Work Identification Information *30*
- Imprint Information *32*
- Copyright Information *32*
- Analytic Information *34*
- Representation Information *34*
- Electronic Citation *35*
- Further Reference Record Codes *35*

2. Humdrum Representation Reference 41

Documentation Style *41*
barlines *43*
**cents *46*
**correl *49*
**date *51*
**dB *56*
**deg *58*

**degree *61*
**diss *64*
**dur *66*
**embel *69*
**freq *73*
**fret *76*
**harm *85*
**IPA *91*
**kern *96*
**melac *107*
**metpos *110*
**MIDI *113*
**mint *116*
**ordo *119*
**pc *121*
**pitch *130*
**recip *134*
**semits *137*
**simil *140*
**solfa *142*
**solfg *145*
**specC *149*
**spect *152*
**synco *154*
**takt *157*
**text *160*
**time *165*
**Tonh *167*
**URrhythm *171*
**vox# *174*
**Zeit *176*

3. Tandem Interpretation Reference 183

Documentation Style *183*
all harmonic intervals *184*
clef *185*
ensemble *187*
expansion lists *188*
fret tuning *190*
harmonic number *192*
instrument *193*
instrument class *199*
key *200*

key signatures *202*
language *204*
meter signatures *206*
MIDI channel *208*
overlay *209*
underlay *209*
section labels *210*
spine path indicators *212*
staff *213*
staff lining *214*
strophe *215*
tempo *217*
thru *219*
timebase *220*
transposition *221*

4. Humdrum Command Reference 225

Command Documentation Style *225*
Inputs and Outputs *228*
Tee *229*
Quotation Marks in the Command Line *230*
Help *230*
On-line Manual *231*
assemble *232*
census *235*
cents *237*
cleave *240*
context *245*
correl *254*
deg *261*
degree *265*
encode *269*
extract *280*
fields *286*
fill *290*
freq *294*
hint *297*
humdrum *303*
humsed *310*
humver *314*
iv *321*
kern *323*
key *328*

melac *332*
metpos *334*
midi *339*
midreset *344*
mint *345*
nf *352*
num *354*
patt *359*
pattern *365*
pc *369*
pcset *373*
perform *376*
pf *381*
pitch *383*
proof *387*
recode *389*
record *394*
regexp *397*
reihe *399*
rend *404*
rid *408*
scramble *411*
semits *415*
simil *418*
smf *425*
solfa *427*
solfg *431*
strophe *434*
synco *441*
tacet *445*
timebase *451*
tonh *453*
trans *457*
urrhythm *462*
veritas *465*
vox *467*
xdelta *469*
yank *476*
ydelta *481*

5. Humdrum Special Files **487**

Documentation Style *487*

6. Regular Expression Reference 491

Regular Expression Documentation *491*
Regular Expressions: A Tutorial Introduction *492*
Literals *492*
Wild-Card *493*
Escape Character *493*
Repetition Operators *493*
Min-Max Character Repetition *495*
Priority of the Longest String Match *496*
Context Anchors *496*
OR Logical Operator *497*
Character Classes *497*
Character Class Keywords *499*
Examples of Regular Expressions *499*
Examples of Regular Expressions in Humdrum *500*
Basic, Extended, and Humdrum-Extended
Regular Expressions *501*
Record-Repetition Operators *502*
regexp *503*

7. Development Reference 509

Command Documentation Style *509*

Bibliography 520

Index 527

Reference Manual

Introduction

The Humdrum Toolkit provides a set of inter-related software tools intended to assist in music research. The toolkit is suitable for use in a wide variety of computer-based musical investigations. The toolkit is less well suited to creative (i.e. generative) musical tasks — such as electroacoustic composition.

Organization of Reference Information

This *Reference Manual* describes all of the individual components included in Release 1.0 of the Humdrum Toolkit. Technical descriptions are given of representation formats, file organizations, command invocations, options, and other information. The *Reference Manual* is divided into seven numbered sections:

Section 1	*General Introduction*
Section 2	*Representation Reference*
Section 3	*Tandem Interpretation Reference*
Section 4	*Command Reference*
Section 5	*Special Files*
Section 6	*Regular Expressions*
Section 7	*Developer's Reference*

Note that numbers identifying each section are given in parentheses in the upper corner of each manual page.

Section 1 *(General Introduction)* provides an overview of Humdrum. This section describes the structural and syntactical elements of Humdrum representations. It also describes the means by which essential bibliographic and reference information is encoded in Humdrum electronic documents.

Section 2 *(Representation Reference)* describes many of the representation schemes currently defined within Humdrum. These schemes include ways of representing period-of-common practice Western music, ways of representing time, pitch, dynamics, tablatures, pitch intervals, functional harmony, embellishment tones, MIDI data, and innumerable other types of music-related information. Note that users are free to define additional or alternative representations that may be better suited to a given application (see Section 7).

Section 3 *(Tandem Interpretation Reference)* describes special-purpose methods for representing various global musical attributes — such as tuning, temperament, instrumentation, transpositions, languages, Da Capo structures, and other general types of information.

Section 4 *(Command Reference)* describes the individual commands contained in Release 1.0 of the Humdrum Toolkit. The majority of these commands accept some sort of musically-pertinent input, and transform this input representation into some other musically-pertinent output representation. Other commands manipulate or modify a given input without changing the form or type of representation.

Section 5 *(Special Files)* describes special files that are created or required by various Humdrum commands.

Section 6 *(Regular Expressions)* describes a common syntax by which complex patterns may be defined. Many of the Humdrum commands rely on regular expressions as a means by which the user can specify some sought pattern. The greater the user's familiarity with regular expression syntax, the more powerful the potential effect of various Humdrum commands.

Section 7 *(Development Reference)* describes how to tailor and augment Humdrum in order to serve specialized research needs. Humdrum defines a format within which an unbounded number of musically-pertinent symbol systems can be defined. Users are thus free to design additional (or alternative) Humdrum representation schemes that are better suited to a given research goal. This section also describes how to write Humdrum compatible software.

Commands and Representations

Several of the entries in different sections of the *Reference Manual* have similar titles — and this can lead to some initial confusion. For example, one entry in Section 2 of the manual is entitled ****semits,** whereas a separate entry in Section 4 is entitled **semits.** The first entry describes a *representation* format for encoding data pertaining to semitone pitch information. By contrast, the entry in Section 4 *(Command Reference)* describes an executable command named **semits.** The **semits** command is able to translate many other types of data into semitone format.

By way of summary, Section 2 describes a set of "objects" or "nouns," whereas Section 4 of the manual describes a set of "actions" or "verbs." In practice, there should be little confusion. The names of Humdrum representations are always designated by two leading asterisks: hence ****solfg** is a data format, whereas **solfg** is the name of a command. In Humdrum, many commands are named according to the type of outputs they produce. So a command named **freq** has a good chance of producing ****freq** as output (although there are some exceptions to this convention).

Documentation Style

Each of the seven sections of the *Reference Manual* begins with a separate introduction describing how the entries within the section are organized. This introductory information outlines the conventions and abbreviations used in the corresponding section of the documentation. The manual entries themselves include a variety of examples that users may find helpful.

The power of the Humdrum Toolkit lies not in the individual commands, but in how these commands can be made to interact. In order to become a proficient user, frequent browsing through the *Reference Manual* is recommended. Even commands that are quite familiar to a user, will often provide options whose utility becomes evident only with experience.

Section 1

Humdrum General Introduction

Introduction

Humdrum is a general-purpose software system intended to assist music researchers in posing and answering research questions. Humdrum's capabilities are quite abstract, and so it is difficult to characterize precisely what it can do. Humdrum can encode information in an unbounded variety of forms. It can transform, classify, coordinate, search, transfer, restructure, contextualize, compare, and otherwise manipulate both pre-defined and user-defined information.†

Humdrum will be of potential benefit to anyone wishing to pursue systematic investigations of musical information. This includes the posing of factual questions about music and the testing of hypotheses about musical organization. Humdrum may thus prove to be of use to music theorists, music analysts, ethnomusicologists, historical musicologists, psychomusicologists, music librarians, dance scholars, linguists, and others.

Although Humdrum facilitates exploratory investigations, Humdrum is best used when the user has a clear problem or question in mind. For example, Humdrum allows users to pose and answer questions such as the following:

- In Bartók, are dissonances more common in strong metric positions than in weak metric positions?
- What passages of the original *Salve Regina* antiphon are preserved in the settings by Tomas Luis de Victoria?
- In Urdu folk songs, how common is the so-called "melodic arch" — where phrases tend to ascend and then descend in pitch?
- Which of the Brandenburg Concertos contains the B-A-C-H motif?
- What are the most common fret-board patterns in guitar riffs by Jimmy Hendrix?
- Which of two English translations of Schubert lyrics best preserves the vowel coloration of the original German?
- Is there evidence of greater metric syncopation in late Mozart than in early Mozart?
- How do chord voicings in barbershop quartets differ from chord voicings in other repertoires?
- After the V-I progression, which harmonic progression is most apt to employ a suspension?
- In what harmonic contexts does Handel double the leading-tone?

---

† Transform (see **humsed**); classify (see **recode**); coordinate (see **assemble**); search (see **grep**, **awk**, **regexp**, **patt**, and **pattern**); transfer (see **rend** and **cleave**); restructure (see **extract**, **yank**, **strophe**, and **thru**); contextualize (see **context**); compare (see **correl** and **simil**).

- Are crescendos in Wagner more strongly associated with rising pitch than is the case for other composers?

Only a few of these questions are *easy* to answer using Humdrum. Although the Humdrum tools may take just minutes to compute an answer for any of the above problems, the primary impediment to a quick solution is the user's skill in interconnecting the right tools for the task at hand. In short, learning Humdrum is akin to learning a programming language.

The Humdrum software system consists of two distinct components: the *Humdrum Syntax* and the *Humdrum Toolkit.* The *Humdrum Syntax* is a grammar for representing sequential symbolic information. The Humdrum syntax is not a single representation scheme; rather, within the syntax, an endless number of representation schemes can be defined. Theoretically, any type of sequential symbolic data may be accommodated — such as square notation, Schenkerian graphs, piano fingerings, changes of emotional states, MIDI data, acoustic spectra, North Indian tabla bols, ballet steps, concurrent television schedules — or even industrial chemical processes.

Within Humdrum's syntactic framework users are free to concoct their own task-specific representations — such as a scheme to represent Telugu notation or Dagomba dance. Humdrum representations may be very highly crafted, or they may be invented in a matter of seconds. It is common to generate intermediate or "throw-away" representations that are used only for a single research task.

The *Humdrum Toolkit* is a set of more than 60 inter-related software tools. These tools manipulate ASCII data conforming to the Humdrum syntax. If the ASCII data represents music-related information, then we can say that the Humdrum tools manipulate music-related information.

The names of some of the Humdrum tools will be readily recognizable by musicians. Humdrum tools such as **degree, key, pitch, record, tacet, trans** and **reihe** may evoke fairly accurate ideas about what they do. Ironically, the most recognizable tools are typically the least useful tools in the toolkit — because they are so specialized. The most powerful Humdrum tools have names such as **cleave, humsed, simil, recode, context, patt** and **yank.**

By themselves, the individual tools of the Humdrum Toolkit are quite modest in their effects. However, the tools are not intended to be self-sufficient. They are designed to work in conjunction with each other, as well as in consort with existing standard UNIX commands. Like musical instruments, their potential usefulness is greatly increased when they are deployed together in various ensembles. Musical problems are typically addressed by linking together successive Humdrum (and UNIX) commands to form one or more command *pipelines.* Although each individual tool may have only a modest effect, the resulting capacity for solving complex problems is legion.

The Humdrum Syntax

The Humdrum syntax provides a framework within which representation schemes can be defined. Each scheme consists of a mapping between the concepts we wish to represent ("signifieds") and how we wish to represent them ("signifiers"). The signifieds are music-related concepts determined by the user, whereas the signifiers consist of the individual ASCII† characters commonly available on computers.

The basis for the Humdrum syntax is a file or group of files. Rather than viewing a file as a linear string of characters, Humdrum regards each file as a two-dimensional plane — much like a sheet of paper. The two dimensions represent *sequence* and *attribute*. *Sequences* of events procede vertically down the page, whereas concurrent *attributes* extend horizontally across the page. Two signifiers that occupy the same horizontal line represent concurrent (or overlapping) events. The basic organization of Humdrum files may be schematically illustrated as follows:

	sequential events ↓			
concurrent events →	A	J	V	etc.
	B	K	W	
	C	L	X	
	D	M	Y	
	etc.	etc.	etc.	

Humdrum encodings consist of a set of one or more lines or *records*. There are three types of Humdrum records:

1. comment records,
2. interpretation records, and
3. data records.

These three record types are mutually exclusive, so it is not possible to mix comments, interpretations, or data records on the same line.

There are two kinds of *comments:* global comments and local comments. *Global comments* may pertain to an entire file (such as the title of a work), whereas *local comments* may pertain to some specific part of the file (such as a particular staff, instrument, note, finger, etc.). Comments are lines that contain an exclamation mark (!) at the beginning of the record (in the first column); subsequent characters up to and including the first occurrence of a carriage return or newline character constitute the comment record. Global comments are denoted by two exclamation marks (!!) at the beginning of the record. Global comments may contain any sequence of printable ASCII characters — including 'blank space' such as tabs and spaces. Local comments may contain any sequence of printable ASCII characters, with the important exception of the tab character. (The reason for excluding tabs in local comments will become clear shortly.) Comments may be used to insert free-format commentaries in Humdrum encodings.

Interpretations are lines that begin with the asterisk character (*). Interpretations are used to identify more precisely the state of the representation — for example, to indicate that an encoded part is for a transposing instrument in E-flat, or to indicate that the representation is for a given Balinese tuning, or that the representation encodes a conductor's physical gestures. Humdrum requires that at least one interpretation must be specified before any data records are encountered. The difference between a comment and an interpretation is that interpretations are

---

† ASCII is an initialism for American Standard Code for Information Interchange.

formal, potentially *executable* statements; interpretations pass information to programs that process the Humdrum encoding.

As in the case of comments, there are two types of interpretations: *exclusive interpretations* and *tandem interpretations*. Exclusive interpretations begin with a double asterisk (**) whereas tandem interpretations begin with a single asterisk (*). Exclusive interpretations are mutually exclusive — only one such interpretation can be active at a given time for a given string of data. No set of data is complete without the presence of an exclusive interpretation. Tandem interpretations, by contrast, provide supplementary information about how a set of data is to be interpreted. Several tandem interpretations may pertain to a given set of data; unlike exclusive interpretations, tandem interpretations are not necessarily mutually exclusive. Various pre-defined Humdrum interpretations are described in detail in Sections 2 and 3 of this document. Section 2 describes a number of exclusive interpretations, whereas Section 3 describes several tandem interpretations.

Lines that do not contain either an exclamation mark or an asterisk in the first column are *data records*. Blank lines (i.e. lines which are either empty, or contain only blank space — such as tabs and spaces) are forbidden in Humdrum. Thus data records may be formally defined as non-empty lines that do not begin with either an exclamation mark or an asterisk. Data records are the work-horses of Humdrum; data records hold the bulk of the encoded information.

In Humdrum, each data record encodes information pertaining to either a particular moment in time or to a particular time window or duration. (Whether a record represents a precise moment or whether it represents an expanse of time depends on the accompanying interpretation.) Each data record may contain one or more data *tokens*. When more than one token is present, tokens are separated from each other by tabs. When several data records are present, multiple tokens will align themselves in columns through the file. Columns have a special importance in Humdrum and are referred to as *spines*.

In a sense, Humdrum data records can be regarded as a two-dimensional table or grid consisting of one or more *spines* and one or more *records* (i.e. columns and rows respectively). By itself, a spine has no particular meaning; it is simply a way of linking together related tokens through time. Spines become meaningful only when they are labelled by adding an interpretation.

By itself, Humdrum recognizes only six ASCII characters. Two of these characters — the exclamation mark (!) and the asterisk (*) — are recognized only when they appear in the first column of a record.† The remaining characters are the period (.), the space, the tab character, and the carriage return (=newline character). As we have seen, the exclamation mark and the asterisk are used to identify comments and interpretations, respectively. The tab and carriage return characters are used to format the data into *spines* and *records*, respectively.

As noted above, the data in the data records are conceptually divided into tokens. In Humdrum, there are two possible types of tokens:

† Or preceded by a tab. See below.

1. *data* tokens, and
2. *null* tokens (.).

Consider, for example, the following file:

```
X	.	X
X	X	X
.	X	X
X	.	X
```

This file consists of three vertical spines and four horizontal records. The first and third spines begin with data tokens, while the second spine begins with a null token. Without the presence of interpretations, the meaning of this file is indeterminate. The file below contains two spines that have been labelled using Humdrum interpretations:

```
**left	**right
X	.
.	X
X	.
.	X
X	.
*-	*-
```

The user has defined two interpretations: "left" and "right." The intention is to represent the footfalls of a person's left and right feet. The above representation simply encodes that the left and right feet have alternating events — such as might be produced by walking or running. Notice that null tokens (.) indicate nothing at all — they merely act as place-holders to maintain the format of the two spines (i.e. the two columns). Notice also that interpretations must be defined for each spine, and that each interpretation consists of some keyword appended to the double asterisks (e.g. `**left`). No intervening spaces are permitted between the interpretation *keyword* (`left`) and the asterisks, however spaces may appear as part of the keyword itself. In addition, when more than one spine is present, both the data tokens and the associated interpretations must be separated by a tab character; spaces cannot be used to separate spines. Finally, note that each spine is formally terminated by a *spine-path terminator* — the asterisk character followed by a minus sign.

Interpretations can be cascaded so that a single spine has more than one interpretation associated with it. This is done through the addition of tandem interpretations. Consider, the following example:

```
**left	**right	**left	**right
*foot	*foot	*arm	*arm
X	.	.	X
.	X	X	.
X	.	.	X
.	X	X	.
X	.	.	X
*-	*-	*-	*-
```

In this case the categories "foot" and "arm" have been added to our representation. The first spine is interpreted both as "left" and as "foot." However, the exclusive interpretation (double asterisks) takes conceptual precedence over the tandem interpretation (single asterisk). That is, tandem interpretations merely modify or supplement the exclusive interpretation. Hence, given the above representation, we would say that "foot" is an attribute of "left" or "right," but we could not say that "left" is an attribute of "foot."

Users are free to define as many different exclusive and tandem interpretations as they wish. For example, a user might define the interpretation `**bowing` that would be suitable for encoding detailed bowing information in works for strings. For each exclusive interpretation, the Humdrum user can re-define the meaning of all of the ASCII characters — with the exception of the tab and the carriage return which always retain their functions as 'token/spine separator' and 'record separator' respectively. The characters: `! . *` can also be re-defined — although there are some restrictions as to how they can be used.†

Since certain types of musical information are quite common, it is helpful to pre-define some general-purpose interpretations. Examples of pre-defined exclusive interpretations include: `**dynam` (dynamics), `**pitch` (pitch), `**harm` (functional harmony), `**pcset` (pitch-class sets), `**degree` (scale degree), `**freq` (frequency), `**mint` (melodic interval), and `**text` (lyrics). Dozens of pre-defined Humdrum representations are described in Section 2 of this manual. However, when browsing through the Humdrum manual, it is important to remember that users are entirely free to concoct their own representations. Humdrum users are not obliged to use any of the pre-defined representations described in this manual. For example, an ethnomusicologist may prefer to define an alternative to the `**text` representation in order to better handle the Inutituuk language. The most important Humdrum software tools will continue to work with any user-defined representations.

The following example illustrates a slightly more complex Humdrum representation. The representation shows one rendering of the opening of "Happy Birthday" — encoded using three interpretations. The `**pitch` interpretation provides a means for representing pitch information, the `**recip` interpretation is able to represent nominal durations, and the `**text` interpretation permits us to represent the lyrics.

† Specifically, the exclamation mark (!) cannot occur in the first column of the record (unless it is used to indicate a comment). Similarly, the asterisk cannot occur in the first column of a record (unless it is used to indicate a Humdrum interpretation). The period (.) character can appear as a data signifier in any situation with the following exceptions: it cannot appear on a line by itself, it cannot appear in the first column of a record if it is followed by the tab character, and it cannot appear at the end of a record if it is preceded by a tab character (unless it is used to encode a null token).

```
!! Mildred and Patti Hill, composers.
!! 'Happy Birthday'
**pitch         **recip      **text
!re A4=440Hz    !            !(lyrics)
G4              8.           Hap-
G4              16           -py
A4              4            birth-
G4              4            -day
C5              4            to
B4              2            you ;
G4              8.           hap-
G4              16           -py
A4              4            birth-
G4              4            -day
D5              4            to
C5              2            you ;
*-              *-           *-
```

The following observations are pertinent. Time moves down the page. Each spine begins with an exclusive interpretation and ends with a spine-path terminator. Spines are separated by tabs. Data records appear only after an exclusive interpretation, but global comments can appear anywhere in the file. Each data record represents concurrent activities across all spines. Global comments ignore the column structure of the spines. However, local comments must conform fully to the spine structure, and can appear only after an exclusive interpretation. Local comments pertain only to their respective spines; the solitary exclamation mark in the second spine is a null comment, which simply acts as a place-holder for the concurrent comments in the other spines. In the case of the `**recip` spine, durations are represented by numbers corresponding to the reciprocals of the American duration names (4=quarter, 2=half; 8.=dotted-eighth). The `**text` representation requires that punctuation marks be isolated from the rest of the data token. When a data token is split in this way, Humdrum requires that the *subtokens* be distinguished by interposing a space character. Data tokens containing multiple subtokens are referred to in Humdrum as *multiple-stops*.

Another Humdrum representation is illustrated by the following excerpt from Fugue 20 from the Well-Tempered Clavier Vol. II. The first and last two measures are shown:

```
!! J.S. Bach, WTC Book II, Fugue 20; BWV 889b
**kern          **kern        **kern
*k[]            *k[]          *k[]
*a:             *a:           *a:
*M4/4           *M4/4         *M4/4
*MM72           *MM72         *MM72
=1              =1            =1
4r              1r            1r
4e              .             .
4c              .             .
4f              .             .
=2              =2            =2
4G#             1r            1r
4r              .             .
8r              .             .
8d              .             .
8B              .             .
8e              .             .
!! Continuing measure 27:
=27             =27           =27
8r              8E            8r
8Dn             8r            32r
.               .             32e
.               .             32f#
.               .             32g#
8BB             4r            8.g#T
8E              .             .
.               .             32f#
.               .             32g#
8C              8r            [2a
8AA             32r           .
.               32B           .
.               32c           .
.               32d           .
8EE             8.c#T         .
8GG             .             .
.               32B           .
.               32c#          .
=28             =28           =28
32FF            8d            8a]
32EE            .             .
8DD             .             .
.               8f            8b
32EE            .             .
32FF            .             .
8.EET           8G#           8e
.               8B            8g#
```

```
32DD            .           .
32EE            .           .
2AAA; 2AA;      2c; 2e;     2a;
==              ==          ==
*-              *-          *-
```

In this example, all of the spines use the **kern exclusive interpretation; **kern is suitable for representing the core pitch and duration information for common-practice musical scores. Several tandem interpretations specify a key-signature having no sharps or flats (*k[]), a key of A minor (*a:), a meter signature (*M4/4), and a tempo (*MM72). (Detailed descriptions of these and other pre-defined tandem interpretations are given in Section 3 of this manual.) Barlines are indicated by an equals-sign. Rests are indicated by the lower-case letter 'r'. In this example, each spine represents a different musical part or voice. Parts are ordered from left-to right with the lowest voice in the left-hand spine. The file is structured like a score turned sideways. In the final chord, double-stops (two pitches) are evident in each of the lower two voices. Notice the use of null tokens (.) as place holders, so when only one voice is active, the other voices merely sustain their previous action. Trills are designated by the upper-case 'T'; tied notes are designated by square brackets; pauses are indicated by the semicolon.

In the **kern scheme, pitches are represented by letter names (a-g and A-G). Lower-case letters indicate pitches above (and including) middle C; upper-case letters indicate pitches below middle C. Letters are repeated (e.g. "cc" or "CC") for each successive octave distance from middle C. In the **kern representation, durations are encoded in the same manner as the previous **recip representation. Although not shown in the above example, **kern is also able to represent phrases, slurs, articulation marks, stem directions, beaming, ornaments, and other features. A complete description of **kern is given in Section 2 of this manual.

Spine Paths

Humdrum representations frequently consist of a fixed number of spines that continue throughout the course of an encoded file. As illustrated above, a typical use of spines might be to encod different "voices" in a musical work. However, there is no reason to equate spines with voices; spines are used for many other purposes. In encoding Humdrum representations it is occasionally useful to be able to vary the number of spines. However, files with varying numbers of spines can pose significant questions of interpretation. Consider, for example, the following sequence of Humdrum-like data records:

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

At the point where three spines is reduced to two spines the continuity is ambiguous: Has spine '3' been discontinued? Or is spine 'B' a continuation of spine '3' with spine 'A' a continuation of spine '1'? For some representations such questions will be of little concern; however in other circumstances the manner in which the spines continue will be of critical importance. For example, if all of the above spines encoded pitch information for various musical parts, a study of melodic intervals would need to resolve the specific melodic paths as the representation moves from three to two spines. Failure to clarify the pitch-paths would make it difficult to determine or search for specific sequences of melodic intervals.

The Humdrum syntax provides special *spine path indicators* that make it possible to resolve such ambiguities and to ensure that the continuity (or lack of continuity) is made clear. Humdrum provides five special path indicators — one of which we have already encountered. Specifically:

- an existing spine may terminate (without continuing further)
- a ne spines may be introduced
- a previous spine may be split into two spines
- two or more spines may be amalgamated into a single spine
- the positions of two spines may be exchanged

Spine path indicators use the following signifiers: the plus sign (add a spine), the minus sign (terminate a spine), the carret (split a spine), the small letter 'v' (join spines), and the small letter 'x' (exchange spines). In addition to these, a *null interpretation* exists — whose purpose is merely to act as a place-holder in interpretation records:

`*-`	terminate a current spine
`*+`	add a new spine
`*^`	split a spine (into two)
`*v`	join (two or more) spines into one
`*x`	exchange the position of two spines
`*`	null interpretation (place holder)

Spine Path Interpretations

Spine paths are types of interpretations, so the spine path indicators are encoded as Humdrum interpretations using the asterisk signifier (*). The following examples illustrate a few possible path changes.

Examples:

```
1   2    3
*   *-   *        (elimination of spine #2)
1   3

1   2    3
*   *x   *x       (exchange spines #2 and #3)
1   3    2

1   2    3
*   *^   *        (splitting of spine #2)
1   2a   2b   3

1   2    3
*   *v   *v       (amalgamation of spines #2 and #3)
1   2&3
```

Notice that in cases where two or more spines are amalgamated, it is imperative that the spines be adjacent neighbors. For example, the arrangement below is forbidden by the Humdrum syntax since it is not clear whether the spines #1 and #3 amalgamate into spine 'A' or spine 'B'.

```
1    2   3
*v   *   *v      (syntactically illegal)
A    B
```

In such cases, amalgamating the two outer spines can be accomplished by first using the exchange path signifier. Here we exchange spines #2 and #3 before amalgamating the original first and third spines:

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

In cases where several amalgamations are necessary, more than one set of interpretation records may be necessary. In the following example, spines #1 and #2 are first joined together (momentarily defining three spines: 1&2, 3, 4). In the subsequent interpretation record, spine #2 (previous spine #3) and spine #3 (previous spine #4) are then joined:

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

In addition, it is possible to join more than two previous spines at the same time:

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

In cases where a new spine is introduced, it is essential to indicate the exclusive interpretation that applies to the new data. Thus an 'add spine' indication must be followed by a second interpretation record:

```
1   2    3
*   *+   *         *    (add a new spine.)
*   *    **inter   *    (define exclusive interpretation for the
1   2    new       3      new spine.)
```

Failing to follow the introduction of a new spine by a subsequent exclusive interpretation is illegal.

The following examples illustrate a variety of more complex path redefinitions:

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

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

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

```
1    2    3    4
*x   *x   *    *
*    *x   *x   *
*    *    *x   *x
2    3    4    1
```

Note that with judicious planning, the user can completely reconfigure all spines within a Humdrum file.

Syntactically, some path constructions are illegal; here are some examples of illegal constructions:

```
1    2    3
*v   *   *v       (The join-spine indication in spine #1 does not
                   adjoin spine #3.)
```

```
1    2    3
*x   *x   *x       (No more than two exchange interpretations at a time.)
```

```
1    2    3
*x   *    *      (Must have two exchange interpretations together.)

1    2    3
*v   *    *      (Must have two or more join interpretations at a
                  time.)

1   2   3
*   *           (Spine eliminated without using a termination
1   2            interpretation.)

1   2   3
*   *   *+       (Adding a new spine should result in 4
1   2   3         interpretations.)

1   2
*   *   *-       (Cannot eliminate non-existent spine.)

1    2
*+   *
1    new   2       (New spine started without specifying new
                    interpretation.)

1   2
*   *+
*   **inter   *      (Interpretation labels the wrong spine.)
A   B         C
```

The Humdrum Syntax: A Formal Definition

With the preceding background it is now possible to define formally a Humdrum representation. First we can define a Humdrum file. A Humdrum file must conform to any one of the following:

1. A file containing *comments, data records* and *interpretations* with the restriction that no data record or local comment appears before the first *exclusive interpretation.*
2. A file containing *data records* preceded by at least one *exclusive interpretation.*
3. A file containing only *comments* and *interpretations* with the restriction that no local comments appear before the first interpretation.
4. A file containing only *interpretations* beginning with an exclusive interpretation.
5. A file containing only global *comments.*
6. A totally empty file (i.e. a file containing no records).

In addition, each spine in a Humdrum file must ultimately end with a path terminator (*-). Only global comments (or new exclusive interpretations) may occur following the termination of all spines. A property of Humdrum files is that the concatenation of two or more Humdrum files will always result in a Humdrum file.

Additional interpretations may be added throughout the file. Global comments may appear anywhere in the file. However, local comments are much more restricted: (1) Local comments may not appear until after the first interpretation record. (2) The number of sub-comments in a local comment record must be equivalent to the number of currently active spines.

Comment	Either a global or local comment.
Global comment	Any record beginning with two exclamation marks (!!).
Local comment	Any record beginning with one and only one exclamation mark (!). Each concurrent spine in the same record must also begin with an exclamation mark.
Null comment	A comment record containing no commentary; only the appropriate exclamation mark(s) present.
Interpretation	Either an exclusive or tandem interpretation. Any record beginning with an asterisk (*).
Exclusive interpretation	Any record beginning with one or more asterisks (*), where at least one spine begins with two asterisks.
Tandem interpretation	Any record beginning with a single asterisk (*) where none of the spines begins with two asterisks.
Path indicator	One of five special interpretations: *+ *- *v *^ *x found only in tandem interpretation records.
Null interpretation	An interpretation for a given spine or spines consisting of just the interpretation signifier (i.e. a single asterisk — *).
Data record	Any record that is not a comment or interpretation. Must contain the same number of tokens as the number of current spines.
Null token	The period (.) either alone on a single record or separated from other characters by a tab. Appears only in data records.
Null data record	A data record consisting only of null tokens.
Spine	A column-like "path" of information — including data records, local comments, and interpretations.

Humdrum Terminology

As a supplement to the above "positive" definition of the Humdrum syntax, we can also describe various inputs that do *not* conform to the Humdrum syntax:

An empty record.
A record containing only tabs.
A record beginning with a tab.
A record ending with a tab.
Any record containing two successive tab characters.
Any data record having fewer or more spines than the immediately preceding data record.
Path indicator having only one join-spine indication
Path indicator having only one exchange-spine indication
Path indicator having more than two exchange-spine indications

Some Illegal Humdrum Constructions

One of the most important commands in the Humdrum Toolkit is the **humdrum** command itself. This command is used to identify whether a file or other input stream conforms to the above Humdrum syntax. Where appropriate, the **humdrum** command issues error messages identifying the type and location of any syntactical transgressions. If no infractions are found, **humdrum** produces no output (i.e. "silence is golden"). All of the remaining commands in the Humdrum toolkit assume that the inputs given to them conform to the Humdrum syntax. Where problems arise, the user should always test the input to assure that it is in the proper Humdrum format.

The examples given below provide further illustrations of Humdrum representations:

```
**form
Introduction
Exposition
Development
Recapitulation
Coda
*-
```

```
**American	**British
quarter	crotchet
eighth	quaver
dotted half	dotted minim
*-	*-
```

```
**Opus/No	**Year
23/1	1821
23/2	1821
23/3	1822?
24	1822
*-	*-
```

```
**recip   **diaton   **accidental   **stem-dir   **kern
4         c          #              /            4c#/
8         d          .              /            8d/
8         e          .              /            8e/
2         f          #              /            8f#/
*-        *-         *-             *-           *-
```

```
**heart-rate
74
73
74
77
78
*-
```

```
**foreground
flute
*^
flute          violin1
*-             *
violin1
*^
violin1        bassoon
*              *^
violin1        bassoon    'cello
*              *          *^
violin1        bassoon    'cello   trombone
*-             *-         *-       *
trombone
*^
trombone       trumpet
*-             *-
```

Section Labels and Repetitions

Musical scores are often notated so as to take advantage of repetitions in the music. Through such devices as repeat marks, Da Capo, Dal Segno, Codas, and other mechanisms, it is possible to represent a musical work in an abbreviated format. Humdrum provides parallel mechanisms that allow works to be represented in succinct ways.

Humdrum files may be logically divided into segments or passages by encoding Humdrum *section labels*. A section label is a type of tandem interpretation that consists of a single asterisk, followed by a greater-than sign, followed by a keyword that labels the section. The following are examples of section labels.

```
*>Coda
*>1st Ending
*>Refrain
```

Notice that spaces can appear in section labels as in `1st Ending`. 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.

Rather than encode multiple copies of a passage, a single instance may be encoded and labelled as a section. The complete version of the work can be reconstructed by referring to an *expansion list.* An expansion list is another tandem interpretation that contains an ordered list of section labels. In effect, the expansion list indicates how the abbreviated file should be expanded to a full-length encoding. Consider the following expansion list:

```
*>[verse1,refrain,verse2,refrain]
```

This list indicates that the abbreviated file contains (at least) three sections, labelled "verse1," "verse2" and "refrain." When the file is expanded, the "refrain" section should be repeated following each verse.

The Humdrum **thru** command expands *abbreviated format* representations to a so-called *through-composed format* in which repeated passages are expanded according to an expansion list. When the **thru** command is invoked, it eliminates any expansion lists present in the input; in addition, **thru** places a `*thru` tandem interpretation in all spines immediately following each instance of an exclusive interpretation in the input. This marks the file as being in a through-composed format. Any other `*thru` tandem interpretations encountered in the input are subsequently discarded. As a result, running a file through **thru** twice will not change the file in any way.

Versions

For works encoded in an abbreviated format, it is not always useful to expand it according to a single fixed recipe. Depending on the performance practice, individual performer, or edition, certain repeats may be avoided, passages may be added, or material eliminated altogether. In short, several different versions or interpretations of the overall organization of a work may exist.

Humdrum provides a mechanism by which several alternative versions of the overall organization of a work may co-exist in the same file. This is achieved simply by encoding more than one expansion list. In order to distinguish different versions, each expansion list is given a unique version label. Consider the following expansion lists:

```
*>Gould82[A,A,B]
*>Landowska[A,A,B,B]
```

Here we see two expansion lists, one labelled `Gould82` and the other labelled `Landowska`. These expansion lists might encode different interpretations of the repeats in a rounded binary form. When the **thru** command is invoked, the user can specify which *version* is intended, and the appropriate through-composed expansion will be output.

Strophic Representations

Section labels and versions allow Humdrum users to select alternative groups of (horizontal) records within a Humdrum file or document. In other circumstances it is useful to be able to select alternative (vertical) paths within a file. Strophic representations may be conceived as "alternative concurrent paths" through a Humdrum document. Examples of alternative concurrent representation paths might include (1) the texts for different verses of a song, (2) alternative renditions of the same passage (such as *ossia* passages), or (3) differing editorial interpretations of a given note or sequence of notes.

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, they all 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 Schoene Muellerin* by Schubert:

```
!! Franz Schubert, 'Das Wandern' from "Die Schoene Muellerin"
**kern          **text
*>[1,1,1,1]     *>[1,1,1,1]
*>1             *>1
*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-
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 '1' 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 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.

The Humdrum **strophe** command can be used to isolate or extract selective strophic data. For example, the user might select certain verses, or extract an *ossia* passage. But before using the **strophe** command, the file must be transformed to a through-composed format via the **thru** command.

The Humdrum Syntax: Conclusion

This introduction to Humdrum has identified the major structural and organizational features of the Humdrum syntax. This syntax provides a framework within which sequential symbolic data can be represented. Individual representation schemes map the ASCII character set (signifiers) to various music-related concepts (signifieds).

Each mapping is designated by an exclusive interpretation. The corresponding data are organized in spines that may meander throughout the file. New spines may be added, spines joined together, exchanged, split, or terminated. Data are organized as tokens — however, the tokens themselves can consist of multiple subtokens separated by spaces. Null tokens can appear as place-holders where no specific data exists.

Free-form comments may be interspersed throughout the file. Global comments pertain to all spines whereas local comments pertain to individual spines. Additional interpretive information may be encoded using tandem interpretations. Both local comments and tandem interpretations may occur anywhere, but must be preceded in the spine by some exclusive interpretation.

Entire passages or sections may be labelled and repetitive material can be eliminated to produce abbreviated formats. Expansion lists indicate the manner by which abbreviated encodings can be restored to through-composed formats. Several alternative versions of the expansion can exist concurrently in a document, and any given named version can be selected for expansion. Alternative concurrent paths of information can be selected in the case of strophically-organized material.

In the ensuing sections of the manual, further details are given concerning specific pre-defined representations. In addition, the operation of the individual Humdrum tools is chronicled in Section 4. In the immediately following discussion, further general information is presented concerning broader reference issues.

Reference Records

An important aspect of any document is the manner by which it is identified and accessed. In the case of printed materials, the author, title, and date of publication provide important pieces of information by which the document can be uniquely identified, retrieved, or cited. In the Humdrum representation syntax, bibliographic or reference information is encoded using specially-formatted global comments called *reference records.* Reference records provide standardized ways of encoding bibliographic information — suitable for computer-based access.

In conjunction with the **grep** and **awk** commands, Humdrum reference records allow users to retrieve electronic documents according to sophisticated search strategies. For example, users can access all works attributed to more than one composer, those works composed in a particular country that exhibit irregular meters, or all works commissioned by a certain individual where the instrumentation includes a specified instrument.

Humdrum reference records are designated by three exclamation marks at the beginning of a record, followed by a three letter code, followed by an optional number, followed by a colon, followed by some text. For example, the following record indicates that the composer of the encoded work is Robert Schumann.

```
!!!COM: Schumann, Robert
```

In most cases, more than one reference code is permitted on the same record; successive codes are then delimited by interposing a tab. For example, the following record indicates both the composer and title of the work.

```
!!!COM: Stevenson, Ronald		!!!OTL: Passacaglia on DSCH
```

A large number of such three-letter codes are pre-defined in Humdrum, and are listed in the ensuing pages. Before identifying specific types of information, we can first identify some general principles for encoding reference information.

Humdrum reference records are intended to serve the international music scholarly community — not just those scholars who speak English. As a general principle, Humdrum reference information is encoded in the original or source language. This typically means that names, titles and other information are encoded in the language of the producer of the work. An exception to this principle arises where the original language does not make use of the Roman alphabet. In these cases, the most scholarly Roman transliteration scheme is used — such as *Pinying* in the case of Mandarin Chinese.

Note that encoding reference information in the source language does not preclude translations of the reference data. Humdrum reference records also provide simple mechanisms for encoding and accessing information in translated forms. Translated reference information may be made to any language for which a Roman transliteration scheme exists. Thus titles for works by Mendelssohn may be rendered in Arabic or Japanese.

Many reference items pertain to the names of people. These include the names of composers, lyricists, librettists, and arrangers. as well as editors, copyright owners, and others. Since Humdrum representations are not limited to the representation of "musical scores," names might include the names of performers, conductors, or even (in the case of perceptual data) listeners. Reference codes that encode people's names follow a common syntax. Most names are in the form of a family-name, followed by a comma, followed by given names, followed by an optional colon, followed by any honorific (such as Dr., von, Sir, etc.). If given names are not known, one or more initials may be encoded where each initial is followed by a period with a single space interposed between successive initials. In the case of Asian names, a comma is placed following the family name. Hence, the Chinese name, Deng Xiao Tie, is represented as Deng, Xiao Tie. Where no family name exists, the name is represented as spoken, as in Guido d'Arezzo, Josquin des Prez, or Chief Falcon Feather. Note that many Javanese-Indonesian names consist of a single name.

In some cases, the most commonly known name is an abbreviation, alias, or stage name — such as Madonna or Liberace. In other cases, the most commonly known names will be corporate names — such as the names of popular groups. Special reference records are provided for these types of reference information and so corporate names should not appear in records intended for proper names. For anonymous individuals, the five-character text string "`Anon.`" should be encoded. Where a specific anonymous individual is understood to be the author, a conventional designation such as "`Anonymous 3`" may be used. Note that Humdrum provides separate reference codes for identifying "attributed composer" and "suspected composer."

Computer-based documents may have complex networks of copyright-related information. Some documents will be encoded from other documents (often printed) that may or may not be in the public domain. The electronic version of the document may have been produced, owned, or licensed by the owner of the original document. In other cases, the original work may be in the public domain, but the edition or arrangement of the transcribed document may be under copyright. Once again, the electronic version of the document may have been produced or licensed by the owner of the original edition or by an arranger.

Unless the electronic document itself was illegally encoded, the document is protected by copyright — normally held by the producer of the electronic edition. In some cases, the producer may elect to assign the ownership to another, to license the ownership to others, to make the document available on a *shareware* basis, or to explicitly place the document in the public domain. Users of electronic media should be aware of the importance of intellectual property, and of the statutory rights of their producers to benefit financially from their labors. Humdrum provides detailed reference capabilities for specifying copyright information. Note that tampering with statements of copyright in an electronic document is a prosecutable offense in most countries.

Over 80 reference codes are pre-defined in Humdrum. Each of these reference records is described below under seven categories: (1) authorship information, (2) performance information, (3) work identification information, (4) imprint information, (5) copyright information, (6) analytic information, and (7) representation information. A final section discusses how to cite electronic documents.

Authorship Information

!!!COM: Composer's name. In some cases, opinions differ regarding the best spelling of a composer's name. If so, all common spellings should be given — each alternative separated from the previous by a semicolon. E.g.

```
!!!COM: Chopin, Fryderyk; Chopin, Frederick
```

With respect to accents, refer to the discussion concerning the **!!!RLN:** reference record (see below). If a work was composed by more than one composer, then each composer's name should appear on a separate **!!!COM:** record with a number designation prior to the colon. For example,

```
!!!COM1: Composer, A.
!!!COM2: Composer, B.
```

!!!COA: Attributed composer. This may include attributions known to be false. Several attributions may be combined on a single record by separating each name by a semicolon. Note that if a document contains both **!!!COA:** and **!!!COM:** records, then the attributed composer is explicitly assumed to be false.

!!!COS: Suspected composer. This reference code indicates the belief of the editor or producer of the document as to the true identity of the composer(s). If more than one composer is suspected, each name should appear on a separate **!!!COS:** record.

!!!COL: Composer's abbreviated, alias, or stage name. e.g. Madonna.

!!!COC: Composer(s) corporate name. Corporate names may include the names of popular groups (especially when the actual composer is not known). Corporate names may also include business names, e.g. Muzak.

!!!CDT: Composer's dates. The birth and death dates should be encoded using the `**Zeit` format described in the *Representation Reference* section of this manual. The `**Zeit` format provides a highly refined representation, including methods for representing uncertainty, approximation, and boundary dates (e.g. prior to ..., after ...).

!!!CNT: Nationality of the composer. This reference information is encoded using the language of the nationality. Thus a German composer is encoded as `Deutscher` rather than "German", and a French composer is encoded as `Francais` rather than "French." Where the composer changed nationality, successive nationalities should be listed (in chronological order) separated by semicolons.

!!!LYR: Lyricist. The name of the lyricist. If more than one lyricist was involved in the work, then each lyricist's name should appear on a separate **!!!LYR:** record with a number designation prior to the colon. If the composer was also the lyricist, this should be explicitly encoding using the independent **!!!LYR:** record — rather than implicitly assumed.

!!!LIB: Librettist. The name of the librettist. If more than one librettist was involved in the work, then each librettist's name should appear on a separate **!!!LIB:** record with a number designation prior to the colon. If the composer was also the librettist, this should be explicitly encoding using the independent **!!!LIB:** record — rather than implicitly assumed.

!!!LAR: Arranger. The name of the arranger. If more than one arranger was involved in the work, then each arranger's name should appear on a separate **!!!LAR:** record with a number designation prior to the colon.

!!!LOR: Orchestrator. The name of the orchestrator. If more than one orchestrator was involved in the work, then each orchestrator's name should appear on a separate **!!!LOR:** record with a number designation prior to the colon.

!!!TXO: Original language of vocal/choral text. The name of the language should be encoded in that language. For example, `russki` rather than `Russian`.

!!!TXL: Language of the *encoded* vocal/choral text. The name of the language should be encoded in the language used for encoding. For example, `Italiano` rather than `Italian`.

!!!TRN: Translator of text. The name of the translator of any vocal, choral, or dramatic text. If more than one translator was involved in the work, then each translator's name should appear on a separate **!!!TRN:** record with a number designation prior to the colon.

Performance Information

Humdrum representations may encode performance-activity information rather than (or in addition to) score-related information. If the representation encodes a given performance (such as a MIDI performance), then the following reference records may be pertinent.

!!!MPN: Performer's name. If more than one performer was involved in the work, then each performer's name should appear on a separate **!!!MPN:** record with a number designation prior to the colon.

!!!MPS: Suspected performer. If more than one performer is suspected, each name should appear on a separate **!!!MPS:** record.

!!!MRD: Date of performance. The performance date should be encoded using the `**date` format described in the *Representation Reference* section of this manual.

!!!MLC: Place of performance. (Local language should be used.)

!!!MCN: Name of the conductor of the performance.

!!!MPD: Date of first performance. The date of first performance should be encoded using the `**date` format described in the *Representation Reference* section of this manual.

Work Identification Information

!!!OTL: Title. The title of the specific section or segment encoded in the current file. Titles must be rendered in the original language, e.g. *Le sacre du printemps.* (Title translations are encoded using other reference records.)

!!!XEN: Translated title (in English). (Note that reference codes are also available for translations to languages other than English, French, German, or Japanese.)

!!!XFR: Translated title (in French). (Note that reference codes are also available for translations to languages other than English, French, German, or Japanese.)

!!!XDE: Translated title (in German). (Note that reference codes are also available for translations to languages other than English, French, German, or Japanese.)

!!!XNI: Translated title (in Japanese). (Note that reference codes are also available for translations to languages other than English, French, German, or Japanese.)

!!!OTP: Popular Title. This reference record encodes well-known or alias titles such as "Pathetique Sonata".

!!!OTA: Alternative title. This reference record encodes earlier or alternate titles.

!!!OPR: Larger (or parent) work from which the encoded piece is a part. For example, "Gute

Nacht" (OTL) from *Winterreise* (OPR).

!!!OAC: Act number. For operas and musicals, this reference record encodes the act number as an Arabic (rather than Roman) numeral. The number may be preceded by the word "Act" as in `Act 3`.

!!!OSC: Scene number. For operas and musicals, this reference record encodes the scene number as an Arabic (rather than Roman) numeral. The number may be preceded by the word "Scene" as in `Scene 3`.

!!!OMV: Movement number. For multi-movement works such as sonatas and symphonies, this reference record encodes the movement number as an Arabic (rather than Roman) numeral. The number may be preceded by the word "Movement" or "mov." etc., as in `mov. 3`.

!!!OPS: Opus number. The number may be preceded by the word "Opus" as in `Opus 23`. Once again, Arabic numerals are used.

!!!ONM: Number. The number may be preceded by the abbreviations "No." or "Nr." as in `No. 4`.

!!!OVM: Volume. The volume number may be preceded by the abbreviation "Vol." as in `Vol. 2`. Arabic numbers are used.

!!!ODE: Dedication. Name of person to whom the work is dedicated. If the work was dedicated to more than one person, then each dedicatee's name should appear on a separate **!!!ODE:** record with a number designation prior to the colon.

!!!OCO: Commission. Name of person or organization that commissioned the work. If the work was commissioned by more than one person, then each commissioner's name should appear on a separate **!!!OCO:** record with a number designation prior to the colon.

!!!OCL: Collector. Name of person who collected or transcribed the work. If the work was collected by more than one person, then each collector's name should appear on a separate **!!!OCL:** record with a number designation prior to the colon.

!!!ONB: Free format note related to the title or identity of the encoded work. If more than one such note is encoded, each should appear on a separate **!!!ONB:** record with a number designation prior to the colon.

!!!ODT: Date of composition. The date (or period) of composition should be encoded using the `**date` or `**Zeit` formats described in the *Representation Reference* section of this manual. The `**date` and `**Zeit` formats provides a highly refined representation, including methods for representing uncertainty, approximation, and boundary dates (e.g. prior to ..., after ...).

!!!OCY: Country of composition. Local names should be used, such as 'Espana'.

!!!OPC: City, town or village of composition. Local names should be used, such as 'Den Haag.'

Imprint Information

!!!PUB: Publication status. This reference record identifies whether the document has ever been "published". One of the following English terms may appear: `published` or `unpublished`.

!!!PPR: First publisher. Name of the first publisher of the work.

!!!PDT: Date first published. The date of publication should be encoded using the `**date` format described in the *Representation Reference* section of this manual.

!!!PPP: Place first published. (Local language should be used.)

!!!PC#: Publisher's catalogue number. This should not be confused with better known scholarly catalogues, such as those of Köchel, Hoboken, etc.

!!!SCT: Scholarly catalogue abbreviation and number. E.g. BWV 551

!!!SCA: Scholarly catalogue (unabbreviated) name. E.g.`Koechel 117`.

!!!SMS: Manuscript source name. For unpublished sources, the manuscript source name.

!!!SML: Manuscript location. For unpublished sources, the location of the manuscript source.

!!!SMA: Acknowledgement of manuscript access. This reference information may be used to encode a free format acknowledgement or note of thanks to a given manuscript owner for scholarly or other access.

Copyright Information

!!!YEP: Publisher of electronic edition. This reference identifies the publisher of the electronic document.

!!!YEC: Date and owner of electronic copyright. This reference identifies the year and owner of the copyright for the electronic document.

!!!YED: Date electronic edition released.

!!!YEM: Copyright message. This record conveys any special text related to copyright. It might convey a simple warning (e.g. "All rights reserved."), convey registration or licensing information, or indicate that the document is shareware.

!!!YEN: Country of copyright. This reference identifies the country in which the electronic document was created, or where the copyright was established. In effect, it identifies the country under whose laws the copyright declaration is to be interpreted.

!!!YOR: Original document. This reference identifies any original source or sources from which encoded document was prepared. Note that original documents may themselves be copyrighted, and that permission may be required in order to create an electronic derivative document. Original documents may also have lapsed copyrights.

!!!YOO: Original document owner. If the electronic document was prepared from a copyrighted original document, this reference identifies the copyright owner of the original document. Note that unless the electronic and original documents have the same owner, some licensing agreement or other legal arrangement is necessary in order to create an electronic derivative document.

!!!YOY: Original copyright year. If the electronic document was prepared from a copyrighted original document, this reference identifies the year of copyright for the original document. Note that some licensing agreement or other legal arrangement is necessary in order to create an electronic derivative document.

!!!EED: Electronic Editor. Name of the editor of the electronic document. If more than one editor was involved in the work, then each editor's name should appear on a separate **!!!EED:** record with a number designation prior to the colon.

!!!ENC: Encoder of the electronic document. This reference identifies the name of the person or persons who encoded the electronic document. (Not to be confused with the electronic editor.) If more than one encoder was involved in the work, then each encoder's name should appear on a separate **!!!ENC:** record with a number designation prior to the colon.

!!!EMD: Document modification description. This record type is used to chronicle all modifications made to the original electronic document. EMD records should indicate the date of modification, the name of the person making the modification, and a brief description of the type of modification made. For each successive modification, a separate **!!!EMD:** record should appear with a number designation prior to the colon.

!!!EEV: Electronic edition version. This reference identifies the specific editorial version of the work. e.g. Version 1.3g Only a single **!!!EEV:** record can appear in a given electronic document.

!!!EFL: File number. Some files are part of a series or group of related files. This record indicates that the current document is file *x* in a group of *y* files. The two numbers are separated by a slash as in:

```
!!!EFL: 1/4
```

!!!EST: Encoding status. This record indicates the current status of the document as it is being produced. Free-format text may indicate that the encoding is in-progress, list tasks remaining, or indicate that the encoding is complete. **!!!EST:** records are normally eliminated prior to distribution of the document.

!!!VTS: Checksum validation number. This reference encodes the checksum number for the file — excluding the **!!!VTS:** record itself. When this record is eliminated from the file, any POSIX.2 standard **cksum** command can be used to determine whether the file originates with the publisher, or whether it has been modified in some way. (See the Humdrum **veritas** command described in Section 4.) Note that this validation process is easily circumvented by malicious individuals. For true security, the checksum value should be compared with a printed list of checksums provided by the electronic publisher.

Analytic Information

!!!AFR: This is a free-form text record that can be used to identify the form (if appropriate) of the work. E.g. fuga, sonata-allegro, passacaglia, rounded binary, rondo.

!!!AGN: This is a free-form text record that can be used to identify the genre of the work. E.g. opera, string quartet, barbershop quartet.

!!!AST: Style, period, or type of work designation. This is a free-form text record that can be used to characterize the style, period, or type of work. This reference can include any term or terms deemed appropriate by the producer of the document. Designations might include keywords or keyphrases such as: Baroque, bebop, Ecole Notre Dame, minimalist, serial, reggae, slendro, heterophony, etc.

!!!ASW: Associated Work. Some works are associated with other works, such as plays, novels, paintings, films, or other musical works. E.g. Shakespeare's Othello. This reference allows associated works to be explicitly identified by author and title.

!!!AMT: Metric Classification. Meters may be classified using combinations of the following keywords: `simple, compound, duple, triple, quadruple, quintuple, irregular.`

!!!AIN: Instrumentation. This reference is used to list all of the instruments (including voice) used in the work. Instruments should be encoded using the abbreviations specified by the `*I` tandem interpretation described in the *Tandem Interpretations* section of this manual. Instrument codes must appear in alphabetical order separated by spaces.† E.g.

```
!!!AIN: clars corno fagot flt oboe
```

Representation Information

!!!RLN: ASCII language setting. This reference identifies the "language" code in which the file was encoded. This is applicable only to computer platforms which provide "extended ASCII" text capabilities (e.g. Danish or Spanish characters).

!!!RDF: User-defined signifiers. All Humdrum representations provide some signifiers (ASCII characters) that remain undefined. Users are free to use these undefined signifiers as they choose. When undefined signifiers appear in a give document, the **!!!RDF:** code should be used to specify what the signifiers denote. E.g.

† Note that alphabetical ordering is essential in order to facilitate searches for specific combinations or subsets of instruments using the **grep** command.

```
!!!RDF1: X=hands cross, left over right
!!!RDF2: x=hands cross, right over left
```

!!!RDT: Date encoded. This reference uses the Humdrum `**date` format to identify the date(s) when the document was encoded.

!!!RNB: Note. This reference provides a free-format text that conveys some document-specific note.

!!!RWG: Warning. This reference may be used to encode explicit warnings concerning the encoded material.

Electronic Citation

Electronic editions of music might be cited in printed or other documents by including the following information. The "author" (e.g. **!!!COM:**), the "title" — either original title (**!!!OTL:**) or translated title (e.g. **!!!OLe:**). The editor (**!!!EED:**), publisher (**!!!YEP:**), date of publication and copyright owner (**!!!YED:**), and electronic version (**!!!EEV:**), In addition, a full citation ought to include the validation checksum (**!!!VTS:**). This number will allow others to verify that a particular electronic document is precisely the one cited. A sample electronic citation might be:

> Franz Liszt, Hungarian Rhapsody No. 8 in F-sharp minor (solo piano).
> Amsterdam: Rijkaard Software Publishers, 1994; H. Vořišek (Ed.),
> Electronic edition version 2.1, checksum 891678772.

In Humdrum files it does not matter where reference records appear. Since it is common for users to inspect the beginning of a file in order to check whether the file is being properly processed, the number of reference records at the beginning of the file should be kept to a minimum. A good habit is to place the composer, title of the work, and copyright records at the beginning of the file, and to relegate all other reference records to the end of the file.

Further Reference Record Codes

The following table provides further pre-defined reference codes not identified in the preceding discussion.

Code	Language
!!!XAL:	translated title in Albanian
!!!XAB:	translated title in Arabic
!!!XAM:	translated title in Armenian
!!!XAZ:	translated title in Azeri

!!!XBE:	translated title in Bengali
!!!XBU:	translated title in Bulgarian
!!!XCB:	translated title in Cambodian
!!!XCA:	translated title in Cantonese
!!!XHR:	translated title in Croatian
!!!XCE:	translated title in Czech
!!!XDA:	translated title in Danish
!!!XNE:	translated title in Dutch
!!!XEN:	translated title in English
!!!XET:	translated title in Estonian
!!!XSU:	translated title in Finnish
!!!XFL:	translated title in Flemish
!!!XFR:	translated title in French
!!!XGA:	translated title in Gaelic
!!!XDE:	translated title in German
!!!XGR:	translated title in Greek
!!!XHB:	translated title in Hebrew
!!!XHI:	translated title in Hindi
!!!XHU:	translated title in Hungarian
!!!XIC:	translated title in Icelandic
!!!XIT:	translated title in Italian
!!!XNI:	translated title in Japanese
!!!XJV:	translated title in Javanese
!!!XKO:	translated title in Korean
!!!XLI:	translated title in Lithuanian
!!!XLA:	translated title in Latin
!!!XLV:	translated title in Latvian
!!!XMG:	translated title in Malayalam
!!!XMA:	translated title in Mandarin
!!!XMO:	translated title in Mongolian
!!!XNO:	translated title in Norwegian
!!!XPL:	translated title in Polish
!!!XPR:	translated title in Portugese
!!!XRO:	translated title in Romanian
!!!XRU:	translated title in Russian
!!!XSR:	translated title in Serbian
!!!XSK:	translated title in Slovak
!!!XSN:	translated title in Slovenian
!!!XES:	translated title in Spanish
!!!XSW:	translated title in Swahili
!!!XSV:	translated title in Swedish
!!!XTA:	translated title in Tamil
!!!XTH:	translated title in Thai
!!!XTI:	translated title in Tibetan

!!!XTU:	translated title in Turkish
!!!XUK:	translated title in Ukranian
!!!XUR:	translated title in Urdu
!!!XVN:	translated title in Vietnamese
!!!XWE:	translated title in Welsh
!!!XHO:	translated title in Xhosa
!!!XZU:	translated title in Zulu

Section 2

Humdrum Representation Reference

Documentation Style

This section of the *Reference Manual* describes various pre-defined representations conforming to the Humdrum syntax. Each representation defines a scheme by which the numbers, letters, and other characters of the ASCII character-set are mapped to musically pertinent signs or "signifieds." Each representation is designated by a different Humdrum *exclusive interpretation*. Exclusive interpretations are denoted by two leading asterisk characters (beginning in the first column of a line or record) followed immediately by an interpretation *keyword*. For any given spine (column of data), only one exclusive interpretation can be active at a given moment, and an interpretation must precede the first data record.

Each documented representation includes a description of the scope of the representation, a complete tabulation of all pre-defined signifiers, and examples of syntactically-correct representations. For many representations, a number of *tandem interpretations* are also pre-defined. Tandem interpretations provide additional contextual information concerning the represented data. In addition, each entry lists all of the existing Humdrum commands that are able to process the representation. Descriptions of the various commands may be found in Section 4 of this *Reference Manual*.

All Humdrum representations can be processed by the general-purpose tools list in the following table.

assemble	paste together Humdrum files
census	determine general properties of a Humdrum file
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

Each reference entry contains information identifying the name and purpose of the representation, a summary description of mappings between signifiers and signifieds, the designated file-type, and a list of Humdrum commands that accept or produce the given representation as input or output. The standard order of documentation sections is as follows: (1) representation, (2) description, (3) file type designation, (4) signifiers used, (5) examples, (6) pertinent commands, (7) tandem interpretations, (8) see also, (9) warnings, (10) limits, (11) note, (12) reference, (13) proposed modifications, and (14) author(s).

REPRESENTATION

barlines — "common system" for representing barlines

DESCRIPTION

Several Humdrum representations employ a common system for representing barlines. This common system is described below. (N.B. This common system is not intended to preclude other ways of representing barlines in Humdrum.)

In the common system, barlines are represented as logical entities, with optional signifiers for specifying the precise visual appearance of the barlines. Barlines are logically signified by the presence of an equals-sign (`=`) in the first column of a spine. Immediately after the equals sign there may follow an optional integer value indicating the measure number (e.g. `=107` — for measure 107). In addition, a lower-case alphabetic character may be appended to the measure number — as in: `=14b`. This convention permits the user to distinguish measure numbers for first and second endings, etc. Measure numbers refer to the measure immediately following the barline, thus the token `=23` occurs just prior to the encoded data for measure 23.

Double barlines are indicated by a minimum of two successive equals signs (`==`). Several consecutive equals signs may be encoded in order to enhance readability (e.g. `=======`).

An additional attribute for barlines is the *pause* — which is represented by the semicolon (;). Thus the token `=4;` means that the barline starting measure number 4 has a pause written above or below it, while the token `=====;` means that a double barline contains a pause indication.

Barlines can be visually rendered in a variety of ways. Where appropriate, users can specify more precisely the notational appearance of the barline by *appending* additional signifiers to the basic signifiers identified above. Barlines may be normal or heavy in width. Barlines of normal width are represented by the addition of the vertical bar (`|`). Heavy barlines are signified by the exclamation mark (`!`). Dotted barlines are signified by the double quote character (`"`). Partial barlines (extending between the second and fourth lines) are signified by the single quote character (`'`). Partial barlines (rendered as a short vertical stroke across the top of the staff) are signified by the single greve character (`` ` ``). "Invisible" barlines are signified by the minus sign (-). In addition, barlines may be associated with repetition marks — pointing left, right, or in both directions. The repeat sign is denoted by the colon character `:`). (Note that this repeat sign is a *visual* signifier, rather than a logical signifier of repetition; see **section labels (3)** and **thru (4)** for information concerning repetition.) All of these visual signifiers may be combined to form complex visual representations, such as a triple barline consisting of a normal line, followed by a heavy line, followed by a normal line, followed by a repeat indicator. See EXAMPLES below.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for “common system” barlines.

<table>
<tr><td>0-9</td><td>measure numbers</td></tr>
<tr><td>a-z</td><td>alternate measures</td></tr>
<tr><td>;</td><td>pause</td></tr>
<tr><td>=</td><td>barline</td></tr>
<tr><td>==</td><td>double barline</td></tr>
<tr><td>|</td><td>normal width visual rendering</td></tr>
<tr><td>!</td><td>heavy width visual rendering</td></tr>
<tr><td>'</td><td>partial barline (from second to fourth line)</td></tr>
<tr><td>`</td><td>partial barline (short stroke at top of staff)</td></tr>
<tr><td>-</td><td>invisible barline</td></tr>
<tr><td>:</td><td>repeat sign</td></tr>
</table>

Summary of `**kern`*-like Barline Signifiers*

EXAMPLES

Several examples of “common system” barlines are given below:

<table>
<tr><td>=</td><td>unnumbered barline</td></tr>
<tr><td>=29</td><td>the beginning of measure 29</td></tr>
<tr><td>=29;</td><td>the beginning of measure 29 with pause</td></tr>
<tr><td>=29a</td><td>first occurrence of measure 29</td></tr>
<tr><td>=29c</td><td>third occurrence of measure 29</td></tr>
<tr><td>=29c;</td><td>third occurrence of measure 29 with pause</td></tr>
<tr><td>==</td><td>double barline</td></tr>
<tr><td>==;</td><td>double barline with pause</td></tr>
<tr><td>====;</td><td>double barline with pause</td></tr>
<tr><td>=|</td><td>unnumbered barline, normal line width</td></tr>
<tr><td>=!</td><td>unnumbered barline, heavy line width</td></tr>
<tr><td>==|!</td><td>double barline, normal line followed by heavy line</td></tr>
<tr><td>=29|</td><td>beginning of measure 29, normal line width</td></tr>
<tr><td>=:|:</td><td>barline with left and right repeats, normal line width</td></tr>
<tr><td>=:||:</td><td>barline with left and right repeats, two normal-width lines</td></tr>
<tr><td>='</td><td>unnumbered barline, rendered with partial barline (mid)</td></tr>
<tr><td>=29`</td><td>beginning of measure 29, rendered with partial barline (top)</td></tr>
<tr><td>=29-</td><td>beginning of measure 29, no barline drawn</td></tr>
<tr><td>==:|!</td><td>double barline with repeat, normal/heavy lines</td></tr>
<tr><td>==|</td><td>logical double barline, visually rendered as single normal line</td></tr>
<tr><td>|</td><td>not a barline</td></tr>
<tr><td>29|</td><td>not a barline</td></tr>
</table>

PERTINENT REPRESENTATIONS

The following Humdrum representations employ "common system" barline signifiers:

****cents**	absolute pitch representation in hundredths of semitones
****deg**	relative scale degree representation
****degree**	absolute scale degree representation
****diss**	sensory dissonance representation
****dur**	event duration (in seconds)
****embel**	representation for embellishment tones
****freq**	frequency representation
****fret**	generalized fretted tablature representation
****harm**	representation for Western functional harmony
****kern**	core pitch/duration of common practice music notation
****melac**	melodic accent representation
****MIDI**	Musical Instrument Digital Interface notation
****mint**	melodic interval representation
****pc**	pitch-class representation
****pcset**	Fortean pitch-class set representation
****pitch**	American National Standards Institute pitch notation
****recip**	beat-proportion representation
****semits**	semitone absolute pitch representation
****solfg**	French solfège (pitch) representation
****specC**	spectral centroid representation
****spect**	discrete frequency spectrum representation
****synco**	represent degree of metric syncopation
****takt**	beat-position representation
****text**	vocal text representation
****time**	relative elapsed time (in seconds)
****Tonh**	German Tonhöhe (pitch) representation
****URrhythm**	represent Johnson-Laird beat prototypes for a passage
****vox#**	representation of number of concurrently active voices

SEE ALSO

humsed (4), **num** (4)

REPRESENTATION

**cents — absolute pitch representation in hundredths of semitones

DESCRIPTION

The **cents representation is used to represent absolute pitch in units of hundredths of semitones with respect to middle C. Each equally tempered semitone spans a distance of 100 cents. Middle C is designated zero cents. All other pitches are represented with respect to this reference, hence A4 is 900 cents and A3 is -300 cents. Cents may be specified as either integer or real values.

Pitch tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The open parenthesis '(' denotes the beginning of a slur. The closed parenthesis ')' denotes the end of a slur. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**cents` spines should be given names with the distinguishing '.cnt' extension.

SIGNIFIERS

The following table summarizes the **cents mappings of signifiers and signifieds.

0-9	decimal values
.	decimal point; or null token
–	minus sign
+	plus sign (optional)
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign

Summary of **cents *Signifiers*

EXAMPLES

A sample document is given below:

```
**cents     **cents
!tempered   !untempered
=1          =1
{1200       {1209.
700         720.4
700         698
=2          =2
(800        (804.1
700)}       722)}
=3          =3
r           r
500 1100    492 1131.2
=4          =4
400 1200    397 1202
==          ==
*-          *-
```

PERTINENT COMMANDS

The following Humdrum commands accept **cents** encoded data as inputs:

cents	change numerical precision of **cents values
freq	translates **cents** to **freq**
kern	translates **cents** to **kern**
pc	translates **cents** to **pc**
pitch	translate **cents** pitch to numerical **pitch**
semits	translates **cents** to **semits**
solfg	translates **cents** to **solfg**
tonh	translates **cents** to **Tonh
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces **cents** data as output:

cents	translates **cents**, **freq**, **fret**, **kern**, **MIDI**, **pitch**, **semits**, **solfg**, **specC**, and **Tonh** to **cents**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **cents:**

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`

Tandem interpretations for ****cents**

SEE ALSO

barlines (2), **cents** (4), ****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)

REPRESENTATION

****correl** — statistical correlation representation

DESCRIPTION

The ****correl** representation is used to characterize the degree of statistical correlation for successive moments. Data tokens for ****correl** decimal values ranging between -1 (perfect negative correlation) and 1 (perfect positive correlation).

FILE TYPE

It is recommended that files containing predominantly `**correl` data should be given names with the distinguishing '.cor' extension.

SIGNIFIERS

The following table summarizes the ****correl** mappings of signifiers and signifieds.

0-9	correlation values specified as integer or real value
.	decimal point; null token
+	plus sign (optional)
–	minus sign

Summary of ****correl** *Signifiers*

EXAMPLES

A sample document is given below:

```
**correl
*C:
*M4/4
.
0.0
+0.3037
+.52
0.6
0.211
.
-.241
-0.17
-1
0.221
.
*-
```

Notice that melodic accent values can be either real or integer values. Rests are represented by the single letter 'r'.

PERTINENT COMMANDS

The following Humdrum command produces ****correl** data as output:

correl	calculate numerical correlation between two spines

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****correl:**

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****correl**

SEE ALSO

barlines (2), **correl** (4), ****simil** (2), **simil** (4)

REPRESENTATION

****date** — absolute time representation (year, month, day, hour, minute, second ...)

DESCRIPTION

The ****date** representation provides a flexible means for specifying particular moments in historical time. The ****date** scheme is able to represent year, month, day, hour, minute, second, and fractional second information. In addition, various degrees of approximation and uncertainty may be represented. N.B. Time-*spans* are not represented by ****date**; see the ****dur (2), **time (2),** or ****Zeit** (2) representations.

In the ****date** representation, date information is encoded according to the following basic syntax:

year/month/day/hour:minute:second.decimal

Date information may be encoded in full, or may consist of isolated elements or parts. The following table shows the most succinct ways of encoding single date values within ****date**:

`.11`	eleven one-hundredths of a second
`11`	11th second
`11:`	11th minute
`11::`	11 o'clock
`11/`	the year 11 A.D.
`/11`	November
`//11`	11th day of the month

Examples of ****date** *components*

Notice that if a single numerical value appears, it is interpreted as *seconds*; if a single value appears followed by a slash, it is interpreted as a *year*; if a single value appears followed by a colon, it is interpreted as a *minutes*. Days and hours require two leading or two trailing delimiters respectively. In general, abbreviated forms of ****date** representations tend to favor the two extremes of time: seconds and years. These are the time frames that are typically of greatest interest to music scholars.

The ****date** representation makes use of the Gregorian calendar and the 24-hour clock; Dates prior to the year 1 A.D. can be specified by prepending the "at" sign (`@`) to the year.

The ****date** representation provides three distinct means for representing approximate moments. It also provides two independent means for representing uncertainty, as well as mechanisms for representing time boundaries (prior to ...; after ...).

If a date token is preceded by the tilde (~) signifier, the entire data token is taken to be *approximate*. Hence, the token `~1556/` represents the approximate year 1556, whereas the token `~1901/9/1/` represents approximately September 1st, 1901. When the lower-case letter 'x' appears in a data token, it indicates that only the adjoining numerical value is approximate. Hence, the token `1921/6x//` means approximately June in the year 1921, whereas the token `1921x/6//` means the month of June in the approximate year 1921. The 'x' may precede or follow a given value — provided it is adjacent to the approximate value.

```
14:30x:
```

means *approximately 30 minutes* after 2 PM, rather than approximately 2:30 PM. In this case, only the value '30' is approximate.

A third method for representing temporal approximation employs the caret (^) to denote a range. For example, the ****date** token:

```
1554/^1557/
```

represents a moment somewhere between the years 1554 and 1557. Conceptually, ****date** is intended to represent *moments* rather than *periods* of time. In other words, the above token represents a moment sometime between 1554 and 1557; it is not intended to signify an event that spanned the years 1554 to 1557. ****date** does not presume how long a moment "lasts," so it is theoretically possible to interpret 1554 to 1557 as a "long moment." However, another representation — ****Zeit (2)** — is intended to represent *periods* of time, and so is better suited to the task of representing an ongoing state or condition. In ****date,** the caret (^) is meant merely to provide a more precise means for specifying the degree of approximation. The following ****date** tokens:

```
1954/6/1/^1954/6/2/
2.23^2.238
```

represent approximate times between June 1st and 2nd, 1954, and between 2.230 and 2.238 seconds respectively.

In addition to methods for representing approximation, ****date** also allows three methods for the encoding of uncertainty. General uncertainty is signified using the question-mark (?). A ****date** token preceded by a question-mark indicates that the date information is *uncertain* (rather than imprecise). For example, the ****date** token:

```
?1661/4//
```

represents the month of April, 1661 — but indicates that this date is uncertain. When the lower-case letter 'z' appears in a data token, it indicates that only the adjoining numerical value is uncertain. This allows the user to be more specific about what aspect of the date is uncertain. For example, a composer's letter might be dated 'September 17th,' but historians may be uncertain of the precise year. This date might be encoded as:

```
1840z/9/17/
```

Note that uncertainty is not the same as approximation. Consider, for example, the contents of a letter dated simply "the 10th" that refers to especially hot summer temperatures. The letter is thought to have been written in 1932:

```
1932z/8x/10/
```

This representation indicates that the year 1932 is uncertain, that the month is approximately August, but that the date is definitely the tenth. As in the case of the 'x' signifier, the 'z' may precede or follow a given value — provided it is adjacent to the uncertain value.

A third method for representing uncertainty is provided by the ****date** logical OR signifier (|). As in the case of the logical BETWEEN (^), the OR signifier separates two component sub-dates. For example, the token `10:41:|11:41:` means "either 10:41 AM or 11:41 AM."

In addition to the approximation and uncertainty signifiers, ****date** also permits the encoding of time boundaries. The less-than sign (<) denotes "prior to" and the greater-than sign (>) denotes "after." For example, the data token `<1100/` means prior to 1100 A.D., whereas the token `>21::` means after 21 hours.

Time boundaries can be mixed with approximation and uncertainty operators. Hence, the data token `<5:30x:` represents a time prior to 5 minutes and approximately 30 seconds. Similarly, time boundaries can be mixed with time ranges. Thus, the data token `<1604/^1605/` means before some time between 1604 and 1605.

Note that > and < apply to entire date tokens (only one of > or < may appear in a given token). The signifiers ? and ~ can apply to subtokens (joined by the *BETWEEN* (^) or *OR* conjunction operators. The signifiers x and z apply to individual numerical values only.

FILE TYPE

It is recommended that files containing predominantly `**data` data should be given names with the distinguishing '.dte' extension.

SIGNIFIERS

The following table summarizes the ****date** mappings of signifiers and signifieds.

<table>
<tr><td>0-9</td><td>decimal values</td></tr>
<tr><td>@</td><td>year B.C. rather than A.D.</td></tr>
<tr><td>/</td><td>year-month, month-day and day-hour delimiter</td></tr>
<tr><td>:</td><td>hour-minute and minute-second delimiter</td></tr>
<tr><td>.</td><td>fractional second delimiter; null token</td></tr>
<tr><td>?</td><td>date uncertain</td></tr>
<tr><td>z</td><td>value uncertain</td></tr>
<tr><td>~</td><td>date approximate</td></tr>
<tr><td>x</td><td>value approximate</td></tr>
<tr><td><</td><td>sometime prior to</td></tr>
<tr><td>></td><td>sometime after</td></tr>
<tr><td>^</td><td>“between” conjunction</td></tr>
<tr><td>|</td><td>“or” conjunction</td></tr>
</table>

Summary of ****date** *Signifiers*

EXAMPLES

Several examples of ****date** data tokens are identified below:

<table>
<tr><th>**date tokens</th><th>meaning</th></tr>
<tr><td>1917/06/02/23:55:00.0</td><td>5 minutes before midnight on June 2nd, 1917.</td></tr>
<tr><td>1917/6/2/23:55:00.0</td><td>5 minutes before midnight on June 2nd, 1917.</td></tr>
<tr><td>///11:59:59</td><td>One second before noon (no day info. provided).</td></tr>
<tr><td>11:59:59</td><td>One second before noon (no day info. provided).</td></tr>
<tr><td>.001</td><td>Time of one millisecond.</td></tr>
<tr><td>23.8</td><td>Time of 23.8 seconds.</td></tr>
<tr><td>/1/1/</td><td>New Years’ Day (January 1st).</td></tr>
<tr><td>1770/</td><td>The year 1770.</td></tr>
<tr><td>1983///.741</td><td>The year 1983; a time-point of 741 milliseconds.</td></tr>
<tr><td><1300/</td><td>Sometime before the year 1300.</td></tr>
<tr><td>>1::</td><td>Sometime after 1 AM.</td></tr>
<tr><td>~14:30:</td><td>Approximately 2:30 PM.</td></tr>
<tr><td>14:30x:</td><td>Approximately 30 minutes past 2 PM.</td></tr>
<tr><td>?14:30:</td><td>Perhaps 2:30 PM.</td></tr>
<tr><td>14:30z:</td><td>Perhaps 30 minutes past 2 PM.</td></tr>
<tr><td>?1848/</td><td>Perhaps 1848.</td></tr>
<tr><td>~1848/</td><td>Approximately 1848.</td></tr>
<tr><td>1847/^1849/</td><td>Sometime between 1847 and 1849.</td></tr>
<tr><td>1847/|1848/|1849/</td><td>Either 1847, 1848, or 1849.</td></tr>
<tr><td>1848/4z//</td><td>1848; perhaps April.</td></tr>
<tr><td>1848x/4//</td><td>Approximately 1848; certainly April.</td></tr>
<tr><td>>~:1:30</td><td>After approximately a minute and a half.</td></tr>
<tr><td>//3/^~//5/3</td><td>Sometime between March and approximately May 3rd.</td></tr>
<tr><td>>?~1933/7/30z/</td><td>Perhaps sometime after about July 30(?) 1933.</td></tr>
</table>

Examples of ****date** *Tokens*

The following is a sample document:

```
**opus   **date
33#1     1864/
33#2     1864/3/31
33#3     1864/
34       1865/4z/
35#1     ?1865/
35#2     <1865/9/
*-       *-
```

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce ****date** as output, or process ****date** encoded data as input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****date**:

meter signatures	`*M6/8`
tempo	`*MM96.3`

Tandem interpretations for ****date**

SEE ALSO

****dur** (2), ****metpos** (2), ****ordo** (2), ****recip** (2), ****takt** (2), ****time** (2), ****Zeit** (2)

REPRESENTATION

****dB** — relative amplitude representation

DESCRIPTION

The ****dB** representation is used to represent relative amplitude in decibels. The reference amplitude is deemed to be 0 dB. In a group of tones, normally the loudest tone is selected as the reference. Typically this means that other tones have negative decibel values.

Relative amplitude values may be either integer or decimal values. Both positive and negative values are permissible.

FILE TYPE

It is recommended that files containing predominantly `**dB` data should be given names with the distinguishing '.db' extension.

SIGNIFIERS

The following table summarizes the ****dB** mappings of signifiers and signifieds.

0-9	decimal values
+	plus sign (optional)
–	minus sign
.	fractional second delimiter; null token

Summary of *****dB*** *Signifiers*

EXAMPLES

The following sample document encodes the relative amplitudes for the first five harmonics of a tone pitched at middle C (C4). Notice the use of the *H tandem interpretations to indicate the harmonic number. In this example, all amplitudes have been given in decibels relative to the first harmonic.

```
**pitch   **dB    **dB    **dB     **dB    **dB
*         *H01    *H02    *H03     *H04    *H05
C4        0.0     -5.3    -24.3    -5.3    -23.7
*-        *-      *-      *-       *-      *-
```

PERTINENT COMMANDS

The following Humdrum command accepts ****dB** encoded data as input:

spect	convert notated sonority to instantaneous spectrum

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****dB:**

MIDI channel	`*Ch1`
harmonic number	`*H3`
meter signatures	`*M6/8`

Tandem interpretations for ****dB**

SEE ALSO

****dynam** (2), ****spect** (2), **spect** (4)

REPRESENTATION

**deg — relative scale degree representation

DESCRIPTION

The **deg representation can be used to represent key-dependent scale-degree information for music in major or minor keys. The **deg representation differs from the related **degree representation in that it encodes relative rather than absolute pitch-height information.

Three types of data tokens are distinguished by **deg: scale degree tokens, rest tokens, and barlines.

Scale degree tokens are encoded as a combination of melodic approach, degree value, and degree alteration. The caret (^) denotes an ascending melodic approach to the current note, whereas the lower-case letter v denotes a descending melodic approach. Repeated pitches carry no melodic approach signifier. The scale degree values are indicated by the numbers 1 (tonic) to 7 (leading-tone). These values may be chromatically altered by raising (+) or lowering (-). The *amount* of chromatic alteration is not indicated; for example, a raised super-tonic is represented as 2+ whereas a doubly-raised super-tonic is also represented as 2+. A lowered submediant is represented as 6-.

Scale degree tokens are always represented with respect to a prevailing major or minor *key*. In the case of minor keys, scale degrees are characterized with respect to the *harmonic minor* scale only. By way of example, the pitch F in the key of A minor is represented as the submediant (6) while F# is represented as the raised submediant (6+). In the same key, G is represented as the lowered seventh (7-) while G#4 is the normal leading-tone (7). In the key of A major, F is represented as the lowered submediant (6-). If this pitch was approached from below, it would be encoded as ^6- — whereas if it was approached from above, it would be encoded as v6-.

Rests are represented by the single letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly **deg data should be given names with the distinguishing '.deg' extension.

SIGNIFIERS

The following table summarizes the **deg mappings of signifiers and signifieds.

0-9	scale degrees, or measure numbers
^	ascending melodic approach
v	descending melodic approach
–	scale degree lowered by one semitone
—	scale degree lowered by two semitones
+	scale degree raised by one semitone
r	rest
=	barline; == double barline

Summary of ****deg** *Signifiers*

EXAMPLES

The sample document given below shows the opening subject of the Fugue in C minor in the second volume of Bach's *Well Tempered Clavier.* The left spine shows a ****kern** encoding while the right spine shows a corresponding ****deg** encoding.

```
!! J.S. Bach, Fugue 2 WTC Book I
**kern          **deg
*M4/4           *M4/4
*c:             *c:
=1              =1
8r              r
16cc            1
16bn            v7
8cc             ^1
8g              v5
8a-             ^6
16cc            ^1
16b             v7
8cc             ^1
8dd             ^2
=2              =2
8g              v5
16cc            ^1
16bn            v7
8cc             ^1
8dd             ^2
16f             v4
16g             ^5
4a-             ^6
*-              *-
```

PERTINENT COMMANDS

The following Humdrum command accepts ****deg** encoded data as inputs:

vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces **deg data as output:

deg	translates **kern, **pitch, **Tonh, **solfg, to **deg

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **deg:

key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for **deg

SEE ALSO

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

REPRESENTATION

****degree** — absolute scale degree representation

DESCRIPTION

The ****degree** representation can be used to represent key-dependent scale-degree information for music in major or minor keys. The ****degree** representation differs from the related ****deg** representation in that it encodes absolute rather than relative pitch-height information.

Three types of data tokens are distinguished by ****degree**: scale degree tokens, rest tokens, and barlines.

Scale degree tokens are encoded as a combination of degree values, degree alterations, and octave designations. The scale degree values are indicated by the numbers 1 (tonic) to 7 (leading-tone). These values may be chromatically altered by raising (+) or lowering (-). The *amount* of chromatic alteration is not indicated; for example, a raised super-tonic is represented as `2+` whereas a doubly-raised super-tonic is also represented as `2+`. A lowered submediant is represented as `6-`.

A second integer value is used to indicate the octave following the ANSI standard pitch designations. For example, the pitch A4 lies in octave 4. (Octaves begin at C and end at B.) In order to avoid confusing scale degrees with octave indications the slash character is used as a sub-token separator. For example, the pitch C4 in the key of C major is represented as `1/4`, while the pitch A#4 in the key of G major is represented as `2+/4`.

Scale degree tokens are always represented with respect to a prevailing major or minor *key*. In the case of minor keys, scale degrees are characterized with respect to the *harmonic minor* scale only. By way of example, the pitch F4 in the key of A minor is represented as the submediant (`6/4`) while F#4 is represented as the raised submediant (`6+/4`). In the same key, G4 is represented as the lowered seventh (`7-/4`) while G#4 is the normal leading-tone (`7/4`). In the key of A major, F4 is represented as the lowered submediant (`6-/4`).

Rests are represented by the single letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**degree` data should be given names with the distinguishing '.dgr' extension.

SIGNIFIERS

The following table summarizes the **degree** mappings of signifiers and signifieds.

0-9	scale degrees, octave designations, or measure numbers
/	scale-degree / octave number separator
–	scale degree lowered
+	scale degree raised
r	rest
=	barline; == double barline

Summary of **degree** *Signifiers*

EXAMPLES

The sample document given below shows the opening subject of the Fugue in C minor in the second volume of Bach's *Well Tempered Clavier*. The left spine shows a **kern** encoding while the right spine shows a corresponding **degree** encoding.

```
!! J.S. Bach, Fugue 2 WTC Book I
**kern          **degree
*M4/4           *M4/4
*c:             *c:
=1              =1
8r              r
16cc            1/5
16bn            7/4
8cc             1/5
8g              5/4
8a-             6/4
16cc            1/5
16b             7/4
8cc             1/5
8dd             2/5
=2              =2
8g              5/4
16cc            1/5
16bn            7/4
8cc             1/5
8dd             2/5
16f             4/4
16g             5/4
4a-             6/4
*-              *-
```

PERTINENT COMMANDS

The following Humdrum commands accept **degree encoded data as inputs:

kern	translates ****degree** to ****kern**
pitch	translates ****degree** to ****pitch**
solfg	translates ****degree** to ****solfg**
tonh	translates ****degree** to ****Tonh**
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces ****degree** data as output:

degree	translates ****kern**, ****pitch**, ****solfg**, ****Tonh**, to ****degree**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****degree**:

key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****degree**

SEE ALSO

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

REPRESENTATION

****diss** — sensory dissonance representation

DESCRIPTION

The ****diss** representation is used to characterize the degree of sensory dissonance for successive acoustic moments. Two types of tokens are recognized by ****diss:** dissonance-tokens and barlines. Dissonance-tokens encode integer values greater than or equal to zero. Larger values represent higher sensory dissonance. Dissonance values reflect the measurement method devised by Kameoka and Kuriyagawa (see REFERENCES).

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**diss` data should be given names with the distinguishing '.dis' extension.

SIGNIFIERS

The following table summarizes the ****diss** mappings of signifiers and signifieds.

0-9	dissonance values specified as integers; measure numbers
=	barline
==	double barline
=;	barline with pause sign

Summary of ****diss** *Signifiers*

EXAMPLES

A sample document is given below:

```
**diss
*C:
*M4/4
=1
65
84
152
160
=2
211
1017
841
1221
=3
*-
```

Note that rests are not represented in the **diss scheme.

PERTINENT COMMANDS

The following Humdrum command produces **diss data as output:

diss calculate the degree of sensory dissonance for successive spectra

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **diss:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for **diss

SEE ALSO

barlines (2), **diss** (4), **spect (2), **spect** (4)

REFERENCES

Kameoka, A. & Kuriyagawa, M. "Consonance theory, part I: Consonance of dyads." *Journal of the Acoustical Society of America,* Vol. 45, No. 6 (1969a) pp.1451-1459.

Kameoka, A. & Kuriyagawa, M. "Consonance theory, part II: Consonance of complex tones and its calculation method." *Journal of the Acoustical Society of America,* Vol. 45, No. 6 (1969b) pp.1460-1469.

REPRESENTATION

****dur** — duration (time-span) representation

DESCRIPTION

The ****dur** representation can be used to encode a sequence of time-spans or successive durations. Units may be seconds, minutes, hours, days, months, and/or years.

In the ****dur** representation, duration information is encoded according to the following basic syntax:

years/months/days/hours:minutes:seconds.decimal

Duration information may be encoded in full, or may consist of isolated elements or parts. The following table shows the most succinct ways of encoding single duration values within ****dur**:

`.11`	11 one-hundredths of a second
`11`	11 seconds
`11:`	11 minutes
`11::`	11 hours
`11/`	11 years
`/11`	11 months
`//11`	11 days

Examples of ****dur** *components*

Notice that if a single numerical value appears, it is interpreted as *seconds*. For example, the data token `32` represents a duration of 32 seconds. If a single value appears followed by a slash, it is interpreted as the number of *years*; if a single value appears followed by a colon, it is interpreted as the number of *minutes*. Days and hours require two leading or two trailing delimiters respectively. In general, abbreviated forms of ****dur** representations tend to favor the two extremes of durations: in seconds and in years.

The data token `1:15:10` represents a duration of one hour fifteen minutes and 10 seconds. It is also possible to encode such durations in seconds alone as in the equivalent — `4510.` If only a single colon is encountered, it is presumed to delineate minutes and seconds as in `5:33` (five minutes and 33 seconds).

The data token `53/` means an elapsed duration of 53 years, whereas `/9//` means a duration of 9 months. The data token `//730/` means 730 days.

The ****dur** representation provides a means for representing approximate durations. It also provides a means for representing uncertainty, as well as mechanisms for representing time boundaries (prior to ...; after ...).

If a duration token is preceded by the tilde (~) signifier, the entire data token is taken to be *approximate*. Hence, the token `~1/` signifies a duration of about 1 year and the token `~3` means a duration of about 3 seconds.

If a duration token is preceded by the question mark (?), the duration is taken to be uncertain. Hence, the token `?3:` signifies a duration of perhaps 3 minutes.

The ****dur** representation does not support the 'x' and 'z' signifies used by related representations such as ****date** and ****Zeit.** However, ****dur** is able to represent shorter-than (<) and longer-than (>) indications. For example, the data token `<1:` means a duration of less than one minute.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**dur` data should be given names with the distinguishing '.dur' extension.

SIGNIFIERS

The following table summarizes the ****dur** mappings of signifiers and signifieds.

0-9	decimal values
/	years-months, months-days and days-hours delimiter
:	hours-minutes and minutes-seconds delimiter
.	fractional second delimiter; null token
?	duration uncertain
~	duration approximate
<	duration shorter than
>	duration longer than
=	barlines
==	double barline

Summary of ****dur** *Signifiers*

EXAMPLES

A sample document is given below:

```
!! Gustav Holst
**dur
*M5/4
=1
.3
.3
.3
1
1.
0.5
0.5
1.0
=2
*-
```

PERTINENT COMMANDS

The following Humdrum command produces ****dur** data as output:

dur change canonical durations (`**recip`) to elapsed time in seconds

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****dur**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for ****dur**

SEE ALSO

barlines (2), ****date** (2), ****metpos** (2), ****ordo** (2), ****recip** (2), ****takt** (2), ****time** (2), ****Zeit** (2)

REPRESENTATION

embel — representation for embellishment tones

DESCRIPTION

The **embel** representation is used to represent the harmonic status of individual pitches in a passage of Western tonal music. There are several different ways of defining and classifying embellishment tones. The **embel** representation categorizes embellishment tones according to the following criteria:

1. whether or not the tone belongs to a given chord
2. whether or not the tone appears in a metrically accented position
3. whether or not the tone is approached by diatonic step
4. whether or not the tone is resolved or left by diatonic step.

Although variations in nomenclature exist, music scholars have identified and defined many types of embellishment tones including: passing tones, neighbour tones, appoggiaturas, escape tones, anticipations, retardations, and suspensions. In addition to these non-chordal embellishment tones, there are also chordal embellishment tones, such as repetitions, and arpeggiations. The following table characterizes each of the various types of embellishment tones according to the criteria outlined above.

Non-Chordal Embellishments:

	Approach	Resolution	Metric position
Accented passing tone	step	step (in same direction)	accented
Unaccented passing tone	step	step (in same direction)	unaccented
Upper neighbor tone	ascending step	step (in opposite direction)	unaccented
Lower neighbor tone	descending step	step (in opposite direction)	unaccented
Accented upper neighbor	ascending step	step (in opposite direction)	unaccented
Accented lower neighbor	descending step	step (in opposite direction)	unaccented
Anticipation	step or leap	same pitch	unaccented
Suspension	same pitch	descending step	accented
Retardation	same pitch	ascending step	accented
Escape tone	step	leap	unaccented
Appoggiatura	leap	step	accented
Changing tone	•	•	•
Pedal tone	•	•	•

Chordal Embellishments:

	Approach	Resolution	Metric position
Repeated chordal tone	same pitch	•	•
Arpeggio tone	leap	step or leap	•

Types of Chordal and Non-chordal Embellishments

When the harmony changes while a pitch is sustained, it is possible for a chordal tone to be transformed into a non-chordal embellishment, such as a suspension. Whenever a tone changes function as an embellishment, at the appropriate point it is indicated by placing the ****embel** data token in square brackets. The square brackets indicate that the note is already sounding (no new note-onset), but has changed function.

Embellishment tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**embel` data should be given names with the distinguishing '.emb' extension.

SIGNIFIERS

The following table provides a complete list of signifiers defined in ****embel**:

aln	accented lower neighbor tone
ant	anticipation
app	appoggiatura
apt	accented passing tone
arp	arpeggio tone (chordal tone)
arp7	arpeggio tone (7th added to the chord)
aun	accented upper neighbor tone
chg	changing tone
cln	chromatic lower neighbor tone
ct	chordal tone (i.e. not an embellishment)
ct7	chordal tone (7th added to the chord)
cun	chromatic upper neighbor tone
cup	chromatic unaccented passing tone
et	escape tone
ln	lower neighbor tone
ped	pedal
rep	repeated tone
ret	retardation
23ret	2-3 retardation
78ret	7-8 retardation
sus	suspension
43sus	4-3 suspension
98sus	9-8 suspension
76sus	7-6 suspension
un	upper neighbor tone
un7	upper neighbor tone (7th added to the chord)
upt	unaccented passing tone
upt7	unaccented passing tone (7th added to the chord)

Summary of ****embel** *Signifiers*

EXAMPLES

A sample document is given below:

```
**kern   **embel   **harm
*C:      *C:       *C:
=1       =1        =1
4g       ct        I
4cc      ct        Ib
8ff      app       Ic
[8ee     ct        .
=2       =2        =2
2ee]     [sus]     V7
4dd      ct        .
=3       =3        =3
4cc      ct        I
*-       *-        *-
```

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce ****embel** as output, or process ****embel** encoded data as input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****embel**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****embel**

SEE ALSO

barlines (2), ****harm** (2)

REPRESENTATION

****freq** — frequency representation

DESCRIPTION

The ****freq** representation can be used to represent frequencies for pure or complex tones. ****freq** distinguishes three types of tokens: frequencies, rests, and barlines. Frequencies are encoded in hertz (Hz) where 440 Hz means 440 cycles per second. Frequencies may be specified as integer or real values (using a decimal). In addition, ****freq** provides limited capabilities for representing phrasing and slurs.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**freq` data should be given names with the distinguishing '.frq' extension.

SIGNIFIERS

The following table summarizes the ****freq** mappings of signifiers and signifieds.

0-9	frequency (in hertz) specified as an integer or real value
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign

Summary of ****freq** *Signifiers*

EXAMPLES

A sample document is given below:

```
**freq   **freq
*pure    *complex
=1       =1
440      (440
440.9    .
880      440
263.     .
.        440)
=2       =2
r        r
==       ==
*-       *-
```

Notice that frequencies can be either real or integer values. Rests are represented by the single letter 'r'.

PERTINENT COMMANDS

The following Humdrum commands accept ****freq** encoded data as inputs:

barks	translates ****freq** to ****barks**
cbr	translates ****freq** to ****cbr**
cents	translates ****freq** to ****cents**
cocho	translates ****freq** to ****cocho**
kern	translates ****freq** to ****kern**
pc	translates ****freq** to ****pc**
pitch	translates ****freq** to ****pitch**
semits	translate ****freq** to numerical ****semits**
solfg	translate ****freq** to numerical ****solfg**
specc	translate ****freq** to numerical ****specC**
tonh	translate ****freq** to numerical ****Tonh**
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces ****freq** data as output:

freq	translates ****cbr**, ****cents**, ****cocho**, ****freq**, ****fret**, ****kern**, ****MIDI**, ****pitch**, ****semits**, ****solfg**, ****specC**, and ****Tonh** to ****cents**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****freq**:

pure tones	`*pure`
complex tones	`*complex`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****freq**

SEE ALSO

barlines (2), ****cbr** (2), ****cents** (2), ****coch** (2), ****degree** (2), ****kern** (2), ****pc** (2), ****pitch** (2), ****semits** (2), ****specC** (2), ****Tonh** (2)

REPRESENTATION

**fret — generalized fretted tablature representation

DESCRIPTION

The **fret representation provides a comprehensive system for representing performance aspects for fretted instruments. The **fret scheme is suitable for representing tablature information for most fretted instruments, including: common 6-string and 12-string guitars, the 4-string bass guitar, lute, mandore, theorbo, chitarrone, mandoline, banjo, dulcimer, and viols. The **fret interpretation is not limited to equal-temperament tuning, and so can be used to represent non-Western fretted instruments, such as the oud and the sitar.

The **fret representation is performance-oriented rather than notationally-oriented. Thus **fret is not suitable for distinguishing different visual renderings — such as traditional French or German lute tablatures. Special-purpose Humdrum representations should be used if the user's goal is to distinguish different forms of visual signifiers.

The **fret representation distinguishes three types of data tokens: tablature-tokens, rests, and barlines. In addition, **fret provides three special-purpose tandem interpretations to specify more precisely the instrument's layout and tuning.

The basic pitches produced by fretted instruments depend on three factors: (1) the relative tuning of the strings with respect to each other, (2) the absolute overall tuning of the instrument, and (3) the position of the frets. Three tandem interpretations allow the user to specify each of these aspects: `*AT:` (absolute tuning), `*RT:` (relative tuning), and `*FT:` (fret tuning). The absolute tuning and fret tuning have default settings,† however, the relative tuning of the open strings (`*RT:`) must be explicitly encoded in a **fret spine.

The absolute tuning of an instrument is indicated by encoding the pitch of the lowest string using the `*AT:` tandem interpretation. For the common six-string guitar, the lowest pitch is normally tuned to E2, and so would be encoded with the tandem interpretation:

```
*AT:E2
```

The `*AT:` interpretation makes use of **pitch-type pitch designations and may also include cents deviation. For example, an instrument tuned 45 cents sharp might be represented as `*AT:E2+45`. Encoding the absolute tuning of an instrument is optional in the **fret representation.

† The default absolute tuning is E2; the default fret tuning is one semitone for each successive fret.

The tandem interpretation (`*RT:`) specifies the relative tuning, number, and arrangement of strings. Some instruments pair strings together in close physical proximity so that two strings are treated by the performer as a single virtual "string." Such paired strings are referred to as *courses*. For example, the 12-string guitar is constructed using 6 courses, and is played much like a 6-string guitar — except that two strings sound together, rather than a single string.

The `*RT:` tandem interpretation encodes the relative tuning of each string by specifying the number of semitones above the lowest string — where each course is delineated by a colon (:). In the following encoding, the relative tuning of successive strings is 0, 5, 10, 15, 19, and 24 semitones above the lowest string. This tuning defines the most common relative tuning for the six-string guitar — where successive strings are tuned above the lowest string by a perfect fourth (5 semitones), a minor seventh (10 semitones), a minor tenth (15 semitones), a perfect fifteenth (19 semitones), and a double octave (24 semitones).

```
*RT:0:5:10:15:19:24
```

For non-Western and other instruments, it is possible to encode non-integer semitone values for various strings, such as a string tuned 9.91 semitones above the lowest string.

When courses consist of more than one string, the relative tuning of each string in the course may differ. When encoding the relative tuning, each string within a course is separated by a comma (,) in the relative tuning interpretation. For example, a 12-string guitar is typically tuned as follows:

```
*AT:E2
*RT:0,12:5,17:10,22:15,27:19,19:24,24
```

In this case, the lower four courses consists of two strings tuned an octave apart, whereas the upper two courses consist of paired unison strings.

Historically, the 6-course *lute* was tuned in a variety of ways — the most common being the so-called *vieil accord* tuning: G2, C3, F3, A3, D4, and G4. From low-to-high, the courses are named bass, tenor, counter-tenor, great mean, small mean, and treble. During the first half of the 16th century, it was common to tune the lower three courses (bass, tenor, counter-tenor) in octaves. Hence, the relative tuning may be represented as follows:

```
*AT:G2
*RT:0,12:5,17:10,22:14,14:19,19:24,24
```

Of course the absolute tuning of the lute was highly variable, so the lowest pitch would not necessarily be G2.

In addition to the absolute and relative tunings, ****fret** also allows the user to specify the tuning of successive frets. In Western instruments, frets are normally placed in semitone increments. For a 12-fret instrument, this semitone arrangement may be explicitly represented using the following tandem interpretation:

```
*FT:1,2,3,4,5,6,7,8,9,10,11,12
```

Each successive numerical value indicates the number of semitones above the open string for successive fret positions. The interpretation begins with the tuning of the first fret rather than the tuning of the open string. The above interpretation is similar to the *default fret tuning* — which is an increase of precisely one semitone for each successive fret. However, the above interpretation specifies only 12 frets, and so fret positions higher than 12 would be considered errors in the above encoding. An instrument constructed with 1/4-tone fret positions can be encoded as follows:

```
*FT:.5,1,1.5,2,2.5,3,3.5,4,4.5
```

The only restriction imposed by `*FT:` is that all strings must have identical fret distances. That is, if the first fret is positioned 1 semitone above the open string, then this relative pitch arrangement must be true of all strings.

Tablature-tokens encode information regarding the fret/finger positions, the manner by which individual strings are plucked (or bowed), pitch-bending, vibrato, damping, harmonics, and other effects. The actions of individual fingers can also be represented. Each tablature-token consists of a several subtokens in the form of Humdrum multiple-stops. Subtokens are delimited by spaces and represent individual courses/strings. A six-string (or six-course) instrument will require six subtokens in each tablature-token.

Subtokens consist of up to five component elements: (1) the string/course status, (2) fret position, (3) bowing/strumming, (4) finger action, and (5) percussive effects. In addition, the tablature-token can encode bowing and strumming information.

In the ****fret** representation, the status of a string/course can occupy one of sixteen states: (1) inactive, (2) plucked, (3) plucked ponticello, (4) plucked sul tasto, (5) plucked tremolo, (6) pizzicato, (7) bowed, (8) bowed ponticello, (9) bowed sul tasto, (10) spiccato, (11) col legno, (12) bowed tremolo, (13) natural harmonic, (14) artificial harmonic, (15) ringing, or (16) damped.

An *inactive* string is signified by the minus sign (-). An ordinary *plucked* string is represented by the vertical line (|). Plucking near the bridge (*plucked ponticello*) is represented by the slash character (/). Plucking near the tone-hole (*plucked sul tasto*) is represented by the back-slash character (\). The repeated *plucked-tremolo* (commonly used on the mandoline) is represented using the octothorpe or hash character (#). *Pizzicato* is represented by the small letter 'z'. Normal bowing of a string is represented by the plus sign (+); *ponticello* bowing is represented by the open parenthesis '(' whereas *sul tasto* bowing is represented by the closed parenthesis ')'. *Spiccato* (bouncing the bow) is represented by the open curly brace '{'. *Col legno* (using the wood of the bow) is represented by the closed curly brace '}'. *Tremolo bowing* is represented by the ampersand (&). *Natural harmonics* and *artificial harmonics* are represented by the lower-case 'o' and upper-case 'O' respectively. String *ringing* is denoted by the colon (:), and the *damping* of a string is denoted by the small letter 'x'.

By way of illustration, the following tablature-token represents a six-string/course instrument, where the first through sixth strings are respectively (1 and 2) plucked, (3) damped, (4) bowed, (5) plucked sul tasto, (6) inactive.

```
| | x + \ -
```

Note that the layout of the strings in a tablature-token always corresponds to the tuning specified in the relative-tuning interpretation. In most representations, the lower-pitched strings will be toward the left side of the tablature token.

Bowing-direction and *strumming* information is prepended to the beginning of the tablature-token. The direction of bowing/strumming is encoded using the left and right angle brackets: > means to bow/strum from the strings on the left side of the representation toward the strings on the right side of the representation. On most instruments this means strumming "downward" — from the lowest- to the highest-pitched strings. The left angle bracket: < means to strum in the opposite direction. A rough indication of the speed of bowing/strumming can be represented by duplicating these signifiers. For example, >> means a slower "downward" bow/strum, and <<< means an especially slow "upward" bow/strum. The percent sign (%) is used to signify the so-called *rasgeado* — or flamboyant Spanish strum. Once again these signifiers appear at the beginning of a tablature-token — whenever they are encoded. Strumming all 6 open strings downward on a commonly-tuned guitar is represented as:

```
*AT:E2
*RT:0:5:10:15:19:24
>| | | | | |
```

Notice that there is no space between the right angle bracket and the first vertical bar.

Fret-position information is indicated through the use of numbers, with the first fret signified by the number '1'. Fret-position numbers are encoded immediately to the right of their respective string/course. For example, the following tablature-token encodes a six-string/course instrument in which the second and third strings are both stopped at the second fret.

```
>| |2 |2 | | |
```

The ****fret** representation also permits the optional encoding of *fingering* information. For the plucking-hand (normally right hand), traditional musical abbreviations are used: *P* (pollex) for the thumb, *I* (index) for the index finger, *M* (medius) for the middle finger, *A* (annularis) for the ring finger, and *Q* (quintus) for the little finger. In addition, the lower-case letter *p* is used to signify the palm of the hand. Note that these letters are applied only to the 'plucking' hand. In the case of the 'fret-board' hand, the lower-case letters *a-e* are used to denote the thumb, index finger, middle finger, ring finger, and little fingers, respectively. Like the fret information, fingering information is encoded immediately to the right of the string to which the information applies. By way of illustration, the finger actions used in the above example may be made explicit as follows:

```
>|P |2bP |2cP |P |P |P
```

The strum is carried out by the thumb, while the index and middle fingers of the fret-hand stop the second and third courses/strings at the second fret. In the following continuation of this representation, the first course/string is replucked by the thumb. With the exception of the second and third courses/strings, the other strings are allows to ring.

```
>|P |2bP |2cP |P |P |P
>|P xIM xIM : : :
```

Notice that in damping the vibrations of the second and third strings, both the index and middle fingers of the 'pluck' hand are used on both strings.

On rare occasions, guitarists will substitute fingers on the fret-board while a string remains sounding. The following example illustrates such a finger-substitution where the middle finger is replaced by the ring finger:

```
| |2b |2c | | |
: :2b :2d : : :
```

Note that in the ****fret** representation, no special signifiers are provided for so-called 'hammer-on' or (ascending-slur), nor for the so-called 'pull-off' or (descending-slur). During the ascending-slur, the sound is produced simply by engaging the next fret. This can be represented in ****fret** by using the "let ring" signifier (:) in conjunction with the appropriate fret notation. The descending-slur can be similarly notated.

The ****fret** representation also provides several short-hand abbreviations for common ornaments and effects. Trills are indicated by the letters 't' (one semitone) and 'T' (two semitones). Mordents are indicated by the letters 'm' (one semitone) and 'D' (two semitones). Inverted mordents are indicated by the letters 'w' (one semitone) and 'W' (two semitones). Turns are indicated by the letters 'S' and '$' (for the inverted "Wagnerian" turn). Two types of vibrato are distinguished: 'v' for transverse vibrato and 'V' for lateral vibrato. Pitch bending is signified by the tilde (~).

Four types of *percussion effects* can be represented using ****fret.** The two most common *tambours* involve tapping on the bridge (represented by the lower-case letter 'u') and tapping on the strings near the bridge (represented by the upper-case letter 'U'). A simple 'tap' on the top-plate is represented by the lower-case letter 'y', whereas a lower-pitched 'thump' on the top-plate is represented by the upper-case letter 'Y'. When sounded alone, these signifiers appear on a record by themselves. When sounded in conjunction with a plucked or (uncommonly) bowed string, these signifiers appear at the beginning of the tablature-token.

Apart from tablature-tokens, ****fret** also permits the encoding of rests and barlines. Rests tokens are denoted simply by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**fret` data should be given names with the distinguishing '.frt' extension.

SIGNIFIERS

The complete system of signifiers used by ****fret** is summarized in the following table.

	Fret-board (left) Hand
1	first fret position
2	second fret position, ...
11	eleventh fret position, etc.
0	open string (not necessarily sounded)
~	bend up in pitch
v	vibrato (transverse)
V	vibrato (lateral)
t	trill (1 fret distance)
T	trill (2 frets distance)
m	mordent (1 fret distance)
D	mordent (2 frets distance)
w	inverted mordent (1 fret distance)
W	inverted mordent (2 frets distance)
S	turn
$	inverted (Wagnerian) turn
a	thumb (of fret hand)
b	index finger (of fret hand)
c	middle finger (of fret hand)
d	ring finger (of fret hand)
e	little finger (of fret hand)
n	no finger (of fret hand)
	Pluck (right) Hand
-	unplucked or unactivated string
|	plucked string (normal)
/	plucked string — near bridge (ponticello)
\	plucked string — near tone-hole (sul tasto)
#	tremolo (plucked, ala mandoline)
z	pizzicato
:	let string ring
x	damp string
o	natural harmonic
O	artificial harmonic
+	bow (normal)
(	bow — near bridge (ponticello)

)	bow — toward fret-board (sul tasto)
{	spiccato
}	col legno (with wood of the bow)
&	tremolo (bowed)
>	strum from low notes to high notes (= down-bow)
<	strum from high notes to low notes (= up-bow)
>>	slower down-strum; slower down-bow
>>	slower up-strum; slower up-bow
>>>	very slow down-strum; very slow down-bow
<<<	very slow up-strum; very slow up-bow
%	rasgeado (Spanish strum)
P	pollex: thumb (of pluck hand)
I	index: index finger (of pluck hand)
M	medius: middle finger (of pluck hand)
A	annularis: ring finger (of pluck hand)
Q	quintus: little finger (of pluck hand)
p	palm (of pluck hand)
N	no finger (of pluck hand)
u	tambour (tap on bridge)
U	tambour (tap on strings near bridge)
y	'tap' on top-plate
Y	'thump' on top-plate

Summary of ****fret** *Signifiers*

EXAMPLES

A sample document is given below. The ****kern** representation echoes the pitches in the ****fret** representation.

```
**recip  **kern    **fret
*        *         *AT:G2
*        *         *RT:0,12:5,17:10,22:14,14:19,19:24,24
*M3/4    *         *M3/4
=1       =1        =1
4        E e g     : |4 : : : |0
8        c         : : : |3 : :
8        d         : : : : |0 x
8        D d e     : |2 : : |2 :
8        f         : : : : |3 :
=2       =2        =2
4        E e g     : |4 : : : |0
4        c         : : : |3 : :
4        c         : : : |3 : x
=3       =3        =3
4        F f a     : |5 : : : |2W
8        f         : : : : |3 :
8        g         : : : : : |0
8        a         : : : : : |2
8        b         : : : : : |4
=4       =4        =4
2        E e cc    : |4 : : : |5v
*-       *-        *-
```

PERTINENT COMMANDS

The following Humdrum commands accept **fret encoded data as inputs:

cents	translates **fret to **cents
freq	translates **fret to **freq
kern	translates **fret to **kern
pitch	translates **fret to **pitch
semits	translates **fret pitch to numerical **semits
solfg	translates **fret pitch to numerical **solfg
tonh	translates **fret pitch to numerical **Tonh

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **fret:

ensemble	`*a2`
absolute tuning	`*AT:`
relative tuning	`*RT:`
fret tuning	`*FT:`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****fret**

SEE ALSO

barlines (2), ****cents** (2), ****freq** (2), ****kern** (2), ****pitch** (2), ****semits** (2), ****solfg** (2), ****Tonh** (2)

REPRESENTATION

****harm** — representation for Western functional harmony

DESCRIPTION

The ****harm** representation provides one method for encoding Western functional harmony. In the ****harm** representation, chords are normally identified within a key context — such as G minor, or A-flat major. Keys are normally indicated using the "key" tandem interpretation. If no key indication is provided, the harmony representation is deemed both key-independent and mode-independent (neither major or minor). (Key-independent representations may be useful for comparing harmonic patterns between groups of works in varying keys.) Changes of key can be defined at any point in a ****harm** representation. The defined key context remains in effect until the occurrence of another key-interpretation or until the key context is specifically "undefined": the reserved key-interpretation `*?:` can be used to "undefine" the key context.

Chords are labelled according to four attributes: (1) chord root, (2) chord type, (3) inversion, and (4) chord alterations.

Chord roots are identified according to the diatonic scale degree. Scale degrees are indicated through the use of Roman numerals: I for tonic, II for supertonic, III for mediant, etc. The specific roots will vary according to whether the key is major or minor. For example, in the key of C major, the III chord will have E as the root, whereas in the key of C minor, the III chord will have E-flat as the root. By definition, the scale degrees of the minor key are assumed to correspond to the pitches of the *harmonic minor* scale. Notice that without a major or minor mode distinction, the roots of the *III* and *VI* chords are ambiguous — they simply denote mediant and submediant chords respectively.

Of course musical passages may contain chords having altered roots. Raising or lowering the root is indicated by prepending a minus sign (-) or octothorpe character (#) respectively. For example, in the major key, a chromatic mediant chord based on the lowered sub-mediant would be encoded as *-VI.* In a notated score, the lowering of the root may be achieved by adding a flat or by adding a natural — depending upon the prevailing key; however, the specific accidental used to lower the root is irrelevant to ****harm.** For example, in the key of C minor, a minor chord having E-natural as a root would be encoded as *#iii.*

In the case of triads, there are four possible chord types: major, minor, diminished, and augmented triads. Upper- and lower-case numerals are used to indicate whether the third of the chord is major or minor. For example, the supertonic chord in a major key would normally be indicated as "ii". In short, major and augmented triads are indicated through a upper-case Roman numeral, whereas minor and diminished triads are indicated through a lower-case Roman numeral. Diminished chords are indicated by the explicit addition of the small letter 'o' — for example, the diminished triad with a root on the leading tone is denoted as "viio." Augmented chords are indicated by the explicit addition of the plus sign

(+) — e.g. "III+" for the augmented triad on the mediant degree (common in the minor key), or "V+" for the augmented altered dominant chord.

In **harm, inversions of chords are indicated using lower-case alphabetic characters: first inversion - "b"; second inversion "c"; third inversion - "d"; etc. Root position is implied, so in the absence of a letter designation (a,b,c ...) the token *IV* means a IV chord in root position. Figured-bass notation is not used in **harm because it proves inconsistent in the spelling of extended tertian chords. In the case of a fully spelled 13th chord in root position, for example, the figured bass would be 1-2-3-4-5-6-7. However, this same figured bass would apply to all inversions of the 13th chord, and so it fails to distinguish any of the possible inversions. In **harm, the first inversion of a 13th chord is signified by the letter "b" whereas the hypothetical 6th inversion of a 13th chord is signified by the letter "f".

Seventh chords are indicated by the addition of the number "7" — as for example in the dominant seventh chord: V7. Ninth, eleventh, and thirteenth chords are similarly represented: e.g. V9, V11, V13. Such extended tertian chords can be encoded in more detail by indicating whether the interval is major or minor — signified by use of the upper- or lower-case letter "M." For example, a dominant minor ninth chord would be represented by "Vm9" whereas a dominant major thirteenth chord would be represented by "VM13". When the interval is not explicitly indicated as major or minor, it is assumed that the actual spelling is in accordance with the prevailing key signature. For example, V9 is equivalent to Vm9 when the prevailing key is minor.

If it is necessary to specify more precisely the actual intervals involved in an extended tertian chord, all intervals may be included: Vm9P11m13. When more than one interval is given, the intervals must be specified in ascending order and must include a major or minor designation. Where intervals are perfect, the upper-case letter "P" is used. Where intervals are augmented, the upper-case letter "A" is used. Where intervals are diminished, the upper-case letter "D" may be used. (It is common practice to represent diminished intervals using the lower-case letter d; in **harm, however, this would be indistinguishable from the designation for third inversion.) Thus the half-diminished seventh chord would normally be represented as viiom7, whereas the full-diminished seventh chord would be represented as viioD7. Doubly-augmented and doubly-diminished intervals can use "AA" and "DD" respectively.

Of course it is rare that a musical passage or work remains within a single key. The use of secondary dominants and modulations requires that some means be provided for indicating shifting key areas. When shifts of key are sanctioned, these should normally be encoded using an "X of Y" approach — e.g. V of V. In the **harm representation, such shifts are indicated via the slash character (/). For example, a dominant seventh chord on the supertonic degree can be represented as V7/ii. If a passage modulates to the subdominant and remains there for some time, chord sequences can be identified as /IV — e.g. V7/IV, I/IV, vi/IV, ii/IV, V/II/IV, II/IV, etc. (Notice the use of I/IV rather than IV; in long sequences of chords it is preferable to encode successive chords within the new key area.) There is no limit to the number of key-area shifts specified in a harmonic token: e.g. V/V/V/V/V/I is syntactically legal.

In traditional harmony, a variety of special chords may be encountered — such as the

Neopolitan chord, and the "ethnic sixth" chords: Italian, French, and German. The Neopolitan chord is a major triad whose root is the lowered supertonic; it is represented in ****harm** by the reserved upper-case letter "N". The Neopolitan chord normally appears as a first inversion chord, so the Neopolitan sixth chord would be represented as "Nb". Notice that the Neopolitan sixth chord is equivalent to "-IIb". The Italian, French, and German augmented sixth chords are represented as "Lt", "Fr", and "Gn" respectively. In addition, the "Tristan chord" (A4m7m10 above bass pitch) has a special designation as "Tr".

Occasionally, chords may appear using *enharmonically* equivalent spelling. Such chords can be encoded by using the enharmonic prefix of the tilde character (~). For example, if a Neopolitan sixth chord is spelled using the raised tonic rather than the lowered supertonic, the chord may be encoded: ~Nb.

In other cases, it may be entirely impossible to identify a chord in terms of traditional Western functional harmony. Such chords may be encoded by specifying a set of intervals above the bass pitch — with the question-mark prefix. For example, in the key of A major, the chord C4, E4, F#4, G#4, D5 can be represented as: ?-IIIM3A4A5M9. Notice that this representation reverts to a descriptive approach and so is no longer truly "functional."

Chord identifications may be characterized as (1) explicit, (2) implied, or (3) alternate. Explicit harmonies occur when most or all of the chordal tones are present. In some cases (such as melodic lines) the harmonies may be implied rather than explicit. Implicit harmonies are indicated by placing the chord signfied in parentheses (). In other circumstances, there will be more than one way of labelling a given harmony. Alternate harmonies are indicated through the use of square brackets []. All other indications are assumed to be explicit. In the case of bi-tonal works, the user may elect to pair explicit and alternate encodings, e.g. iii[v/vi], or make use of two independent ****harm** spines. Two or more ****harm** spines may be necessary in the case of polytonal works.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**harm` data should be given names with the distinguishing '.hrm' extension.

SIGNIFIERS

The following table summarizes the ****harm** mappings of signifiers and signifieds.

-	lowered root
#	raised root
+	augmented triad
o	diminished triad
I	chord degree (major)
i	chord degree (minor)
V	chord degree (major)
v	chord degree (minor)
b	first inversion chord
c	second inversion chord
d	third inversion chord
e	fourth inversion chord (ninth chords)
f	fifth inversion chord (eleventh chords)
g	sixth inversion chord (thirteenth chords)
r	rest
7	added seventh
9	added ninth
11	added eleventh
13	added thirteenth
.	null token
/	secondary function, e.g. V "of" vi
m	minor interval
M	major interval
P	perfect interval
A	augmented interval
D	diminished interval
AA	doubly-augmented interval
DD	doubly-diminished interval
Nb	Neopolitan sixth chord
N	Neopolitan chord in "root position"
Lt	Italian augmented sixth chord
Fr	French augmented sixth chord
Gn	German augmented sixth chord
Tr	Tristan chord
~	enharmonically-spelled chord
( )	implicit harmony
[ ]	alternative functional harmony label
viiom7	half-diminished seventh chord
viioD7	full-diminished seventh chord

Summary of ****harm** *Signifiers*

EXAMPLES

A sample document is given below:

```
**harm
*C:
!! An example.
=1
I
IVb
.
V7/V
=2
I/V
vib/V[iiib]
ii/V[vi]
viioD7/V
=3
V[I/V]
.
IV
Nb
V7d
=4
!! Minore
*c:
i
?-IIIM3A4A5M9
#iiiob
IV
*?:
Lt
=5
ic
Vm9
I
==
*-
```

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce ****harm** as output, or process ****harm** encoded data input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****harm**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****harm**

SEE ALSO

barlines (2), ****embel** (2), ****kern** (2)

REPRESENTATION

****IPA** — representation for International Phonetic Alphabet

DESCRIPTION

The ****IPA** scheme provides a means for representing phonetic information via the International Phonetic Alphabet (IPA). This representation is a Humdrum adaptation of the ASCII transliteration scheme for IPA developed by Evan Kirshenbaum. The ****IPA** representation permits the encoding of the basic phonetic constituents of spoken or sung utterances for most languages.

Two types of data tokens are distinguished in ****IPA**: phonetic segments and silences.

Spans of silence are represented by the percent sign (%) appearing as an isolated token.

Phonetic segment tokens typically encode syllables or single-syllable words. Each syllable consists of one or more successive phonemes, where each phoneme is signified by a single character with an optional modifier. The beginning of the token may include one of three stress indicators. A stressed syllable is signified by the apostrophe ('). A less stressed syllable is signified by the comma (,). The absence of either the apostrophe or the comma indicates an unstressed syllable.

In the case of tonally inflected languages (such as all Chinese dialects), numbers may follow indicating the type of tonal inflection. For example, in Mandarin, the number 1 indicates a high tone, the number 2 indicates a rising tone, the number 3 indicates a falling-rising tone, and the number 4 indicates a falling tone.

Individual phonemes are represented according to the signifiers listed in the table given below. For example, the upper-case letter 'A' signifies the 'aw' vowel as in the standard American pronunciation of *law*, *cot* and *bother*. Any vowel or diphthong signifier can be modified by the subsequent presence of a tilde; the modified vowel is pronounced with open nasal passages — as in the French "un bon vin blanc," which is encoded as:

```
**IPA
W~
bo~
va~
blA~
*-
```

In the case of consonants, two modifiers are possible. Any consonant signifier that is followed by a semicolon indicates that the front of the tongue is positioned as at the beginning of the word *yacht*. Any consonant signifier that is followed by a dash indicates that the consonant and the preceding *schwa* (see below) are pronounced as an independent syllable — as in the case of the words *battle*, *mitten* and *eaten*.

FILE TYPE

It is recommended that files containing predominantly `**IPA` data should be given names with the distinguishing '.ipa' extension.

SIGNIFIERS

The following table summarizes the ****IPA** mappings of signifiers and signifieds.

@	**schwa**†; as in (unaccented) *banana, collide, alone* or (accented) *humdrum*
V	**schwa** (IPA symbol: ʌ); as in the British pronunciation of *hut*
R	**R**‡; as in *burn, operation, dirt, urgent*
&	short **a** (IPA symbol: æ); as in *mat, map, mad, gag, snap, patch*
A	**ä** (IPA symbol: *a*); as in *bother, cot*, and, with most American speakers, *father, cart*
a	**å**; *father* as pronounced by speakers who do not rhyme it with *bother.*
E	short **e** (IPA symbol: ε or E); as in *get, bed, peck, edge*
i	long **e** (IPA symbol: ē); as in *beat, greed, evenly, easy*
I	short **i** (IPA symbol: ɪ or ɩ); as in *tip, banish, active*
O	**ȯ** (IPA symbol: o or upside-down 'c'); as in *law, all, shawm*
W	**oe** digraph (IPA symbol: œ); as in the French *boeuf*, German *Hölle*
u	**ü**; as in *rule, youth, few, ooze*
U	**u̇** (IPA symbol: ʋ or ᴜ or ω); as in *pull, wood, book*
y	**ue**; as in the German *füllen, hübsch*, or French *rue*
vowel~	following a vowel* indicates a vowel or diphthong pronounced withopen nasal passages; as in the French "un bon vin blanc" (`W~ bo~ va~ blA~`)
b	**b** (IPA symbol: b or ƀ); as in *baby, cabin, rob*
d	**d**; as in *deed, dulcimer, ader*
f	**f**; as in *feel, cuff*, forte
g	**g**; as in *go, bag, gift*
h	**h**; as in *hear, ahead, horn*
k	**k**; as in *cook, take, ache*
x	**K** (IPA symbol: k̠); as in the German *ich, Buch*
l	**l**; as in *libretto, lily, pool*
m	**m**; as in *music, limb, nymph*
n	**n**; as in *ino, nstrument, own*
N	**eng** (IPA symbol: 'n' with a tail); as in *sing, finger, ink*
p	**p**; as in *piano, beeper, lip*
r	**r**; as in *reed, organ, car*
s	**s**; as in *seek, source, bass*
S	**sh** ["esh"] (IPA symbol: ʃ); as in *shy, crescendo, special*

† (All footnotes appear at the end of this table.)

t	**t**; as in *tempo, tie, attacca*
T	**th** ["thorn"] (IPA symbol: θ); as in *thin, path, ether*
D	**th** ["eth"] (IPA symbol: ð); as in *then, either, smooth*
v	**v**; as in *voice, vivid, live*
w	**w**; as in *we, away*
j	**j**; as in *yes, young, cue, onion*
z	**z**; as in *zone, raise, xylophone*
Z	**zh** ["yogh"§]; as in *measure, vision, azure*
consonant-	following a consonant (l-, n-, m-, or N-)** indicates a consonant preceded by a *schwa* that is pronounced as an independent syllable; as in *battle, mitten, eaten*
consonant;	following a consonant,†† indicates that the front of the tongue is positioned as in the beginning of the word 'yard'
^	preceding phoneme is palatalized
´	primary stress (should precede stressed sound)
,	secondary stress (should precede stressed sound)
%	silence signifier

Summary of ****IPA** *Signifiers*

† The IPA *schwa* is notated as an upside-down 'e'.

‡ The IPA symbol consists of a *schwa* with a hook.

§ The IPA *yogh* is written like a flat-topped number '3' that has been lowered in height.

* In IPA such vowels are marked by the presence of a tilde above the vowel.

** In IPA such consonants are marked by the presence of a vertical bar below the consonant.

†† The IPA symbol consists of a superscript letter 'j' either following or hooked beneath the consonant.

EXAMPLES

Sample syllables and their corresponding ****IPA** encodings are given below:

```
**text   **IPA
book     bUk
sing     sIN
tag      t&g
now      nAU
loud     lAUd
out      AUt
```

```
site    saIt
side    saId
buy     baI
job     dZAb
gem     dZEm
edge    EdZ
join    dZOin
judge   dZ@dZ
day     deI
fade    feId
date    deIt
cape    keIp
youth   juT
few     fju
mute    mjut
cue     kju
cure    kjUr
coin    kOin
troy    trOi
bone    boUn
know    noU
beau    boU
chin    tSIn
*-      *-
```

The following example encodes a sentence as might be spoken by American and British speakers.

```
**text   **IPA       **IPA
*        *American   *British
I        aI          aI
hear     hir         hiV
the      D@          DI
sec-     'sEk        'sEk
-re-     rI          r^
-ta-     ,t&         tri
-ry      ri          .
*-       *-          *-
```

PERTINENT COMMANDS

The following Humdrum commands accept ****IPA** encoded data as inputs:

formant identifies first two formant frequencies for common vowels

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****IPA**:

instrument	*I
instrument class	*IC
meter signatures	*M6/8
tempo	*MM96.3

Tandem interpretations for ****IPA**

SEE ALSO

formant (4)

LIMITS

The International Phonetic Alphabet itself is not well suited to the representation of clicks commonly found in African languages, and this limitation is evident in the ****IPA** representation. A more precise phonetic representation also developed by Evan Kirshenbaum might be adapted as the basis for a more refined Humdrum representation.

REFERENCES

This representation is a Humdrum adaptation of the ASCII transliteration scheme for IPA developed by Evan Kirshenbaum.

REPRESENTATION

****kern** — core pitch/duration representation for common practice music notation

DESCRIPTION

The ****kern** scheme can be used to represent basic or core information for period-of-common-practice Western music. The ****kern** representation allows pitch and canonical duration information to be encoded. In addition, ****kern** also provides limited capabilities for representing accidentals, articulation, ornamentation, ties, slurs, phrasing, barlines, stem-direction and beaming. In general, ****kern** is intended to represent the underlying semantic information implied by a musical score rather than the visual or orthographic information embodied by a given printed rendition; ****kern** is designed to facilitate analytic applications rather than music printing or sound generation. Other Humdrum representations should be used for these latter purposes.

Three types of data tokens are distinguished in ****kern**: notes, rests, and barlines.

Notes can encode a variety of attributes including absolute pitch, accidental, canonical duration, articulation, ornamentation, ties, slurs, phrasing, stem-direction and beaming.

Pitches in ****kern** are encoded as "nominally" equally-tempered values. Transposing instruments are represented at concert pitch with a tandem interpretation indicating the nature of the transposition. Pitch information is encoded through a scheme of upper- and lower-case letters. Middle C (C4) is represented using the single lower-case letter `c`. Successive octaves are designated by letter repetition, thus C5 is represented by `cc`, C6 by `ccc` and so on. For pitches below C4, upper-case letters are used: `C` for C3, `CC` for C2, and so on. This same scheme is used for other pitch letter-names. Changes of octave are deemed to occur between B and C. Thus the B below middle `c` is represented as `B`; the B below `CC` is represented as `BBB`, and so on.

Accidentals are encoded using the octothorpe (#) for sharps, the minus sign (-) for flats, and the lower-case letter `n` for naturals. Accidentals are encoded immediately following the diatonic pitch information. Double-sharps and double-flats have no special representations in ****kern** and are simply denoted by repetition `(##)` and `(--)`. Triple- and quadruple accidentals are similarly encoded by repetition. Sharps, flats, and naturals are mutually exclusive in ****kern,** so tokens such as `cc#n` and `GG-#` are illegal. In addition, natural signs may not be repeated (i.e. `nn`).

In ****kern,** all pitches are encoded as contextually independent absolute values. Pitches must be encoded with the appropriate accidental, even if the accidental is specified in a key-signature, or is present earlier in the same measure. Transposing instruments must be notated at (sounding) concert pitch (although see the tandem interpretation **transposition (3)**).

Note tokens may be modified by the presence of additional signifiers.

The ****kern** representation provides no generic means for representing "curved lines" found in printed scores. Lines must be explicitly interpreted as ties, slurs or phrases. The open brace `{` denotes the beginning of a phrase. The closed brace `}` denotes the end of a phrase. The open parenthesis `(` and closed parenthesis `)` signify the beginning and end of a slur respectively. The semicolon `;` denotes a pause. The open square bracket `[` denotes the first note of a tie. The closed square bracket `]` denotes the last note of a tie. The underscore character `_` denotes middle notes of a tie.

Additional signifiers are provided for denoting articulation marks and ornaments. The letters `T` and `t` are used to signify whole-tone and semitone trills, respectively. Whole-tone and semitone mordents are signified by the letters `M` and `m`. Inverted mordents are signified by `W` (whole-tone) and `w` (semitone). (Note that trills, mordents, and inverted mordents wider than two semitones in size are also denoted by the upper-case signifier.) The letter `S` signifies a turn, whereas the dollar sign (`$`) signifies an inverted (or Wagnerian) turn. When a concluding turn is appended to the end of an ornament (such as a trill), the upper-case letter `R` is added to the ornament signifier (as in `tR` and `TR`). In addition to these ornaments, ****kern** provides a signifier for (multi-note) arpeggiation (`:`). The presence of ornaments other than trills, mordents, inverted mordents, and turns types can be indicated by the generic ornament symbol (`O`).

Articulation marks include the apostrophe (`'`) for staccato, the double-quote (`"`) for pizzicato, the greve (`` ` ``) for attacca, the tilde (`~`) for tenuto, and the caret (`^`) for all note-related accents (including < and >). In addition, ****kern** provides signifiers for up-bow (`v`) and down-bow (`u`). The presence of other articulation types can be indicated by the generic articulation symbol (`I`).

As noted, the ****kern** scheme is intended for analytic applications rather than as a means for representing visual renderings of notation. Nevertheless, ****kern** distinguishes up-stems, down-stems, and beamings in order to assist in analytic tasks such as the determination of voicings and in order to facilitate the parsing of note-groupings. Up-stems and down-stems are indicated by the slash (/) and backslash characters (\) respectively. The beginning and ends of beams are signified by the upper-case letters `L` and `J`. Multiple beams are indicated via letter repetition, e.g. `LL` ⇔ `JJ` and `LLL` ⇔ `JJJ` for double- and triple-beams respectively. Partial beams may extend to the right (`K`) or left (`k`). Again, multiple partial beams are indicated via letter repetition. By way of example, a doubly-dotted sixteenth note beamed to a thirty-second note can be represented as:

```
16..LL
32JJkk
```

Slurs and phrase markings can be nested (e.g. slurs within slurs) and may also be elided (e.g. overlapping phrases) to a single depth. Nested markings mean that one slur or phrase is entirely subsumed under another slur or phrase. For example: `(     (     )     )` means that a short slur has occurred within a longer slur. Elisions are overlaps, for example, where an existing phrase fails to end while a new phrase begins. In ****kern** the ampersand character is used to mark elided slurs or phrases. For example: `{     &{     }     &}` means that two phrases overlap — the initial phrase ending after second phrase has begun.

Durations are encoded in a manner identical to the ****recip** representation. Durations are encoded as nominal proportions using integer numbers and the period character. With the exception of the value zero, durations are represented by reciprocal numerical values corresponding to the American duration names: `1` for whole note, `8` for eighth, `32` for thirty-second, etc. The number zero (`0`) is reserved for the *breve* duration (i.e. a duration of twice the length of a whole note). Dotted durations are indicated by adding the period character (`.`) immediately following the numerical value — hence `8.` signifies a dotted-eighth note and `2..` signifies a doubly-dotted half note. Any number of augmentation dots may follow the duration integer.

Triplet and other irregular durations are represented using the same reciprocal logic. Three quarter triplets in the time of four quarters (a whole duration) are signified by the value `3` (i.e. "third notes"). Eighth-note quintuplets (5 in the time of 4) are represented by the value `10` (a half duration divided by 5). See ****recip (2)** for further details.

The ****kern** representation also allows for the encoding of acciaccaturas, non-canonical groupettos, and appoggiaturas. Depending on the expected analytic application, one way to handle these notational devices is to encode the notes according to the manner in which they are typically performed. Alternatively, since these notes are viewed as embellishment notes and hold potentially less analytic status, a special designation for these notes can be useful for certain types of studies.

Acciaccaturas (grace notes) are visually represented as minature notes denoted by a slash mark through the stem. In ****kern** these notes are treated as "durationless" notes and are designated by the lower-case letter `q`. Hence, the token `G#q` denotes a G#3 acciaccatura. Non-canonical groupettos are minature (non-cue) notes (typically in groups) whose stems do not contain a slash, and whose notated durations cause the total notated duration for the measure to exceed the prevailing meter. These groupetto notes are encoded as notes having their notated durations, but are also designated by the upper-case letter `Q`. Hence, a minature sixteenth-note middle C would be encoded as `16cQ`. Depending on the analytic task, these notes may be treated as equivalent to their notated durations, or they may be discarded. For example, the **timebase** command eliminates these notes. Note that data records containing acciaccaturas or groupettos notes must not include normal notes.

In the case of appoggiaturas, ****kern** requires that they be encoded as performed. An appropriate duration is assigned to the appoggiatura according to common performance practice. The duration of the subsequent note is reduced by a corresponding amount. The status of the two notes forming the appoggiatura is nevertheless marked. The appoggiatura itself is designated by the upper-case letter `P`, whereas the subsequent note (whose notated duration has been shorted) is designated by the lower-case letter `p`.

Rests tokens are denoted by the single lower-case letter `r` along with a numerical duration signifier. Rests may also have the attributes of stem-direction, beaming, slur, and phrase, but rests cannot be assigned articulation or ornamentation attributes.

Barlines are represented using the "common system" for barlines — see **barlines (2)**. A barline is denoted by the occurrence of an equals sign (=) at the beginning of the token — followed by an optional measure number (integer) followed by an optional letter (single lower-case alphabetic character), followed by an optional pause marking (`;`). Double-barlines are represented by the occurrence of two or more equals signs (==) at the beginning of the token — followed by an optional pause marking (`;`).

In representing any work, editorial interpretations are inevitable. It may be necessary to make explicit certain implicit information in a score (such as expanding abbreviations), or it may be necessary to estimate missing or unreadable information. Interpreting the voicings (that is, making explicit the degree of connectedness between successive pitches) is an important editorial function in ****kern** representations. The ****kern** representation provides several special-purpose signifiers to help make explicit various classes of editorial amendments, interpretations, or commentaries. Five types of editorial signifiers are made available: (1) *sic* (information is encoded literally, but is questionable) signified by `Y`; (2) *invisible symbol* (unprinted note, rest or barline, but logically implied) signified by `y`; (3) *editorial interpretation,* (a "modest" editorial act of interpretation — such as the interpretation of accidentals in *musica ficta*) signified by `x`; (4) *editorial intervention* (a "significant" editorial intervention) signified by `X`; (5) *footnote* (accompanying local or global comment provides a text commentary pertaining to a specified data token) signified by `?`.

FILE TYPE

It is recommended that files containing predominantly `**kern` data should be given names with the distinguishing '.krn' extension.

SIGNIFIERS

The following table summarizes the ****kern** mappings of signifiers and signifieds.

0	breve duration
1	whole duration
2	half duration
3	half-note triplet duration
4	quarter duration
6	quarter-note triplet duration
8	eighth duration
12	eighth-note triplet duration
16	sixteenth duration
24	sixteenth-note triplet duration
32	thirty-second duration
64	sixty-fourth duration
128	one-hundred and twenty-eighth duration
.	duration augmentation dot (must follow a number)
-	flat sign (minus character)
--	double-flat (two successive minus characters)

a-g	absolute pitches above middle C
A-G	absolute pitches below middle C
#	sharp
##	double sharp
h	end glissando
j	harmonic
k	partial beam extending leftward
kk	two partial beams extending leftward
m	mordent (semitone)
n	natural sign
p	designator of a note subsequent to an appoggiatura
q	acciaccatura (grace note signifier; in lieu of duration)
r	rest
t	trill (semitone)
u	down-bow
v	up-bow
w	inverted mordent (semitone)
y	editorial mark: invisible symbol; unprinted note, rest or barline, but logically implied
H	begin glissando
I	generic articulation (unspecified articulation)
J	end beam
JJ	end two beams
K	partial beam extending rightward
KK	two partial beams extending rightward
L	start beam
LL	start two beams
M	mordent (whole tone)
O	generic ornament (unspecified ornament)
P	appoggiatura note designator
Q	groupetto note designator
R	signified ornament ends with a turn
S	turn
$	Wagnerian turn
T	trill (whole tone)
W	inverted mordent (whole tone)
x	editorial interpretation; immediately preceding signifier is interpreted
xx	editorial interpretation; entire data token is interpreted
X	editorial intervention; immediately preceding signifier is an editorial addition
XX	editorial intervention; entire data token is an editorial addition
Y	editorial mark: *sic* marking; information is encoded literally, but is questionable
z	sforzando

<space>	(space character) multiple-stop conjunction — indicates joint note-tokens
=	barline; == double barline
[	first note of a tie
]	last note of a tie
_	middle note(s) of a tie (underscore)
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign
'	staccato mark
s	spiccato
"	pizzicato mark
`	attacca mark
~	tenuto mark
^	accent mark
:	arpeggiation (of multi-note chord)
,	breath mark
/	up-stem
\\	down-stem
&	elision marker (for slurs or phrases)
?	editorial mark: immediately preceding signifier has accompanying editorial footnote
??	editorial mark: entire preceding data token has accompanying editorial footnote

Summary of **kern *Signifiers*

CONTEXT DEPENDENCIES

In general, signifiers in the **kern representation are intended to be context independent. This means, for example, that the data tokens `{(16ff#/'` and `/ff#16'({` are equivalent. A few exceptions to this principle are necessary in order to maintain the meaning of multiple-character signifiers.

Numbers encoding a duration must be contiguous. That is, a sixteenth note may be encoded as `16ff#` or `ff#16` but not as `1ff#6`. Augmentation dots (signified by the period) must follow immediately after the associated duration numerals. Thus `16.ff#` is acceptable, but not `16ff#.` or `.16ff#`. Sharps, flats, and naturals must follow immediately after the corresponding alphabetic pitch signifiers (`16ff#` but not `16#ff`). Signifiers that can be repeated must be contiguous. This include pp, PP, LL, JJ, XX, xx, ??, ##, --, and ..

The elision marker (`&`) must immediately precede the associated slur (`&{ ... &}`) or slur (`&( ... &)`).

Barlines follow a strict contextual syntax. Barlines must begin with one or more equals-signs, followed by an optional measure number, followed by an optional lower-case letter, followed by an optional pause signifier.

In certain applications, it may be necessary to have a canonical ordering of the signifiers within **kern data tokens. For example, when comparing two ostensibly identical **kern files, trivial differences of signifier orderings will cause UNIX commands such as **cmp** and **diff** to declare the files to be "different." In this case, it is useful to adopt a standard order of signifiers so that direct file comparisons may be made. Similarly, differences in signifier orderings can cause problems for pattern matching tasks. For example, in searching for a sixteenth-note F-sharp, it is convenient to define a simple regular expression — such as `16f#` rather than having to define a regular expression that handles all possible contextual orderings — such as `(16.*f#)|(f#.*16)`. For this reason, a canonical ordering of the **kern signifiers is given in the following table.

signified	signifier(s)	comments
1. open phrase elision indicator	**&**	must precede {
2. open phrase mark	{	
3. open slur elision indicator	**&**	must precede (
4. open slur	(	
5. open tie	[	
6. duration	**0123456789**	any combination; signifiers may be repeated
7. augmentation dot(s)	**.**	signifier may be repeated
8. pitch	**abcdefgABCDEFG**	only one of; signifier may be repeated
9. accidental	- or # or **n**	- and # may be repeated
10. pause	;	
11. ornament	**MmS$TtWwR** or **O**	**O** precludes others; no repetition of a given signifier; must appear in order given
12. appoggiatura designator	**p** or **P**	
13. acciaccatura designator	**q**	
14. groupetto designator	**Q**	
15. articulation	**z** ' " ‘ ~ ^ : or **I**	**I** precludes others; no repetition of a given signifier; must appear in order given
16. bowing	**u** or **v**	only one of
17. stem-direction	/ or \	only one of
18. beaming	**L** or **J**	signifiers may be repeated
19. partial beaming	**k** or **K**	signifiers may be repeated
20. user-defined marks	**il**	one or more of;
	NUVZ	may be repeated but
	@ % + | < >	must be in order given
21. closed or continuing tie	] or _	
22. closed slur elision indicator	**&**	must precede)
23. closed slur	)	
24. closed phrase elision indicator	**&**	must precede }
25. closed phrase mark	}	
26. breath mark	,	
27. editorial marks	**xx** or **XX**	

Canonical Ordering of Signifiers in ****kern** *Note Tokens*

Note that the editorial signifiers `?`, `y`, and `Y`, as well as the single editorial signifiers `x` and `X` (as opposed to `xx` and `XX`) can appear anywhere in a data token, except as the first character.

EXAMPLES

A sample document is given below:

```
!! J.S. Bach, Fugue 2 WTC Book I
!! (3 parts), in c minor; BWV 847b
**kern       **kern       **kern
*M4/4        *M4/4        *M4/4
*MM72        *MM72        *MM72
*k[b-e-a-]   *k[b-e-a-]   *k[b-e-a-]
*c:          *c:          *c:
=1           =1           =1
1r           8r           1r
.            16cc         .
.            16bn         .
.            8cc          .
.            8g           .
.            8a-          .
.            16cc         .
.            16b          .
.            8cc          .
.            8dd          .
=2           =2           =2
1r           8g           1r
.            16cc         .
.            16bn         .
.            8cc          .
.            8dd          .
.            16f          .
.            16g          .
.            4a-          .
.            16g          .
.            16f          .
=3           =3           =3
1r           16e-         8r
.            16cc         .
.            16bn         16gg
.            16an         16ff#
.            16g          8gg
.            16fn         .
.            16e-         8cc
.            16d          .
.            8c           8ee-
.            8ee-         16gg
.            .            16ff#
.            8dd          8gg
.            8cc          8aan
=4           =4           =4
*-           *-           *-
```

PERTINENT COMMANDS

The following Humdrum commands accept **kern encoded data as inputs:

census -k	determine general characteristics of a **kern file
cents	translates **kern to **cents
deg	translates **kern to **deg
degree	translates **kern to **degree
freq	translates **kern to **freq
hint	calculate harmonic intervals from **kern input
key	estimate the key of a **kern input
mint	calculate melodic intervals from **kern input
pc	translates **kern to **pc
pitch	translates **kern to **pitch
proof	check for errors in **kern encoded file
semits	translate **kern to numerical **semits
solfa	translate **kern to numerical **solfa
solfg	translate **kern to numerical **solfg
synco	measure degree of metric syncopation
timebase	reformat **kern score with constant timebase
tonh	translate **kern to numerical **Tonh
trans	transpose **kern score
urrhythm	characterize the rhythmic prototypes in a passage
vox	determine active and inactive voices in a Humdrum file

The following Humdrum commands produce **kern data as outputs:

kern	translates **cents, **degree, **freq, **fret, **MIDI, **pitch, **semits, **solfg, **specC, and **Tonh to **kern
timebase	reformat **kern score with constant timebase
trans	transpose **kern score

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **kern:

clef	*clefG2
instrument	*I
instrument class	*IC
key signatures	*k[f#c#]
key	*c#:
meter signatures	*M6/8
tempo	*MM96.3
timebase	*tb32
transposing instrument	*ITr

Tandem interpretations for ****kern**

SEE ALSO

barlines (2), ****cents** (2), ****degree** (2), ****freq** (2), ****fret** (2), ****MIDI** (2), ****mint** (2), ****pc** (2), ****pitch** (2), ****recip** (2), ****semits** (2), ****solfg** (2), ****specC** (2), ****Tonh** (2)

REPRESENTATION

**melac — melodic accent representation

DESCRIPTION

The **melac** representation is used to characterize the degree of pitch-related stress (melodic accent) for successive moments. Three types of tokens are recognized by **melac**: stress-tokens, rests, and barlines. Stress-tokens encode decimal values ranging between 0 (no stress) and 1 (maximum stress).

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**melac` data should be given names with the distinguishing '.mac' extension.

SIGNIFIERS

The following table summarizes the **melac** mappings of signifiers and signifieds.

0-9	melodic accent values specified as integer or real value; measure numbers
r	rest
=	barline; == double barline
=;	barline with pause sign

Summary of **melac** *Signifiers*

EXAMPLES

The following sample document shows a single spine of **melac** data for the bass line of J. S. Bach's two-part keyboard Invention No. 1. The corresponding **kern** and **semits** representations are also shown:

```
!! J.S. Bach, Two-part Invention 1 (BWV 772)
**kern          **semits          **melac
*C:             *C:               *C:
*M4/4           *M4/4             *M4/4
=1              =1                =1
2r              r                 r
16r             r                 r
16C             -12               1
16D             -10               0.33
16E             -8                0.221
16F             -7                0.556
16D             -10               0.121
16E             -8                0.241
16C             -12               0.121
=2              =2                =2
8G              -5                0.241
8GG             -17               0.17
4r              r                 r
16r             r                 r
16G             -5                1
16A             -3                0.33
16B             -1                0.221
16c             0                 0.556
16A             -3                0.121
16B             -1                0.241
16G             -5                0.121
=3              =3                =3
*-              *-                *-
```

Notice that melodic accent values can be either real or integer values. Rests are represented by the single letter 'r'.

PERTINENT COMMANDS

The following Humdrum command produces ****melac** data as output:

melac calculate melodic accent values for successive pitches

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****melac**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****melac**

SEE ALSO

barlines (2), **melac** (4), ****semits** (2), **semits** (4)

REPRESENTATION

**metpos — position in metric hierarchy

DESCRIPTION

The **metpos representation consists of simply a set of numerical values indicating the order of events. Normally, a **metpos spine simply encodes a set of ascending integers marking the succession of events:

```
**metpos
1
2
3
4
5
*-
```

The **metpos representation also recognizes **kern-like barlines. The presence of an equals-sign (`=`) in the first column of a **metpos spine is used to denote a barline. Immediately after the equals sign there may follow an optional integer value indicating the measure number (e.g. `=107` — for measure 107). In addition, a lower-case alphabetic character may be appended to the measure number — as in: `=14b`. This convention permits the user to distinguish measure numbers for first and second endings. Measure numbers refer to the information immediately following the barline, thus the token `=23` occurs just prior to the encoded beats for measure 23.

Double barlines are indicated by using two or more successive equals signs (`==`). Several consecutive equals signs may be encoded in order to enhance readability (e.g. `=======`). An additional attribute for barlines is the *pause* — which is represented by the semicolon (;). Thus the token `=4;` means that the barline starting measure number 4 has a pause written above or below it, while the token `=====;` means that a double barline contains a pause indication.

FILE TYPE

It is recommended that files containing predominantly `**metpos` data should be given names with the distinguishing '.mtp' extension.

SIGNIFIERS

The following table summarizes the **metpos mappings of signifiers and signifieds.

0-9	decimal values
.	fractional delimiter; null token
=	barlines
==	double barline

Summary of ****metpos** *Signifiers*

EXAMPLES

A sample document is given below:

```
**kern   **metpos
*M4/4    *M4/4
*tb16    *tb16
*c:      *
=1       =1
8r       1
.        5
16cc     4
16bn     5
8cc      3
.        5
8g       4
.        5
8a-      2
.        5
16cc     4
16b      5
8cc      3
.        5
8dd      4
.        5
=2       =2
*-       *-
```

PERTINENT COMMANDS

The following Humdrum commands accept ****metpos** encoded data as inputs:

synco	measure degree of metric syncopation
urrhythm	characterize the rhythmic prototypes in a passage

The following Humdrum command produces ****metpos** data as output:

metpos	generate metric position values for timebase-formatted ****kern** or ****recip** inputs

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****metpos**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for ****metpos**

SEE ALSO

barlines (2), ****date** (2), ****dur** (2), ****kern** (2), **kern** (4), **metpos** (4), ****ordo** (2), ****recip** (2), ****takt** (2), ****time** (2), **timebase** (4), ****Zeit** (2)

REPRESENTATION

****MIDI** — Musical Instrument Digital Interface notation

DESCRIPTION

The ****MIDI** representation is a Humdrum version of the well-known MIDI standard. MIDI is an industry standard used to exchange information between sound synthesizers. MIDI is also used in various software applications such as some music printing software. MIDI is a type of "tablature" notation. It describes a set of performance actions rather than specifying the sounded result. MIDI represents note-on, note-off, information, for various "channels." MIDI events include note-on, note-off, key-velocity, after-touch, control codes, and system-exclusive codes. MIDI does not represent many other musically-pertinent signifiers, such as ties, slurs, phrasings, ornaments, etc. MIDI does not represent rests.

FILE TYPE

It is recommended that files containing predominantly `**MIDI` data should be given names with the distinguishing '.hmd' extension.

SIGNIFIERS

The following table summarizes the ****MIDI** mappings of signifiers and signifieds.

0-9	decimal values
/	value delimiter
=	barline; == double barline
–	note off

Summary of ****MIDI** *Signifiers*

EXAMPLES

A sample document is given below:

```
!! C-major scale.
**MIDI
*Ch8
72/60/64
72/-60/64 72/62/64
36/-62/64 36/64/64
36/-64/64 36/65/64
36/-65/64 36/67/64
36/-67/64 36/69/64
36/-69/64 36/71/64
36/-71/64 36/72/64
72/-72/64
*-
```

Each ****MIDI** data token consist of three elements or components. Each element is an integer value; elements within a data token are delimited by the slash character (/).

The first element in a data token represents the number of clock ticks (since the previous event) before the event is to occur. The absolute duration of a single clock tick is determined by the MIDI clock speed, which is variable.

The second element in a data token represents the MIDI key number — that is, the address of the key event. Key events can be either *key-on* or *key-off*. Key-on events are represented by positive integers, whereas key-off events are represented by negative integers. For example, -60 means to turn-off key 60.

The third element in a data token represents the MIDI key-velocity value. MIDI instruments normally interpret key-velocity as dynamic or accent information. Higher key-velocity values are associated with accented notes. Permissible key-velocity values range between 0 and 127. encodings. In the case of key-off events, the key-velocity component represents the after-touch.

Note that the key-velocity component of a data token is optional and need not appear. However, both the clock-tick value and the key-event values must be present in each ****MIDI** data token.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

PERTINENT COMMANDS

The following Humdrum commands accept ****MIDI** encoded data as inputs:

cents	translates ****MIDI** to ****cents**
fade	fade-in or fade-out ****MIDI** data

freq	translates ****MIDI** to ****freq**
kern	translates ****MIDI** to ****kern**
perform	play Humdrum ****MIDI** files
pitch	translates ****MIDI** to ****pitch**
semits	translates ****MIDI** to ****semits**
smf	generate standard MIDI file
solfg	translates ****MIDI** to ****solfg**
tonh	translates ****MIDI** to ****Tonh**

The following Humdrum commands produce ****MIDI** data as outputs:

midi	produces ****MIDI** output from ****kern** input
record	records ****MIDI** data from a MIDI input

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****MIDI**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`

Tandem interpretations for ****MIDI**

SEE ALSO

barlines (2), ****cents** (2), ****freq** (2), ****kern** (2), ****pitch** (2), ****semits** (2), ****specC** (2), ****Tonh** (2)

REPRESENTATION

****mint** — melodic interval representation

DESCRIPTION

The ****mint** representation provides a means for characterizing sequential (melodic) pitch distances. The ****mint** representation permits the encoding of four types of data tokens: interval tokens, pitch offsets, rests, and barlines.

Interval tokens consist of up to three component parts: (1) interval direction, (2) diatonic interval size, and (3) interval quality. The interval direction is signified by a leading plus sign (+) for ascending motion, or a leading minus sign (-) for descending motion. No special signifier is provided to denote unison motion. Note that interval direction information is optional, and so is not a mandatory aspect of the ****mint** representation.

The diatonic interval size is signified by integer values — 1 for unison, 2 for second, 3 for third, 11 for eleventh, and so on.

Interval qualities are signified as follows: the upper-case letter 'M' for major intervals, lower-case 'm' for minor intervals, upper-case 'P' for perfect, lower-case 'd' for diminished, upper-case 'A' for augmented. Doubly-diminished and doubly-augmented intervals are represented by 'dd' and 'AA' respectively. Triply- and quadruply- diminished or augmented intervals are similarly represented by character repetition, e.g. 'AAA'. Note that interval quality information is optional, and so is not a mandatory aspect of the ****mint** representation.

The normal or canonical order for data elements in a ****mint** interval token is as follows: (1) direction of interval motion, (2) diatonic interval size, (3) interval quality. Note that more than one interval token may appear within Humdrum multiple-stops.

Pitch offsets indicate initial or starting absolute pitches from which successive pitch intervals arise. Pitch offsets are optional and need not appear in a ****mint** representation. Pitch offsets are distinguished by square brackets. Within the square brackets appears a pitch designation using any one of the following pitch-related representations: ****kern**, ****pitch**, ****solfg** or ****Tonh**. For example, the following ****mint** pitch offsets are considered equivalent: 'b' (****kern**), 'B5' (****pitch**), 'si5' (****solfg**), 'H5' (****Tonh**). Where necessary, several pitch offsets may appear as a Humdrum multiple-stop.

Rests are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**mint` data should be given names with the distinguishing '.mnt' extension.

SIGNIFIERS

The following table summarizes the **mint** mappings of signifiers and signifieds.

0-9	interval size signifiers; measure numbers
A	augmented interval quality
d	diminished interval quality
M	major interval quality
m	minor interval quality
P	perfect interval quality
r	rest
=	barline; == double barline
–	descending interval
+	ascending interval
;	measure pause

Summary of **mint** *Signifiers*

EXAMPLES

The following example shows a `**pitch` spine on the left and a corresponding `**mint` spine on the right.

INPUT	OUTPUT
!! Wagner, Tristan Prelude	!! Wagner, Tristan Prelude
**pitch	**mint
*M6/8	*M6/8
A3	[A3]
=1	=1
F4	+m6
.	.
E4	-m2
=2	=2
F3 B3 D#4 G#4	-M7 -P4 -m2 +M3
*-	*-

PERTINENT COMMANDS

The following Humdrum command produces **mint** data as outputs:

mint produces **mint** output from **pitch**, **kern**, **solfg**, or ****Tonh** input

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****mint**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****mint**

SEE ALSO

barlines (2), ****hint** (2), **hint** (4), **kern** (2), **mint** (4), **solfg** (2), ****Tonh** (2), **xdelta** (4), **ydelta** (4)

REPRESENTATION

****ordo** — sequential order of events representation

DESCRIPTION

The ****ordo** representation consists of a set of numerical values indicating the order of events. An ****ordo** spine simply encodes a series of ascending integers marking the succession of events:

```
**ordo
1
2
3
4
5
*-
```

The ****ordo** representation recognizes "common system" barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**ordo` data should be given names with the distinguishing '.ord' extension.

SIGNIFIERS

The following table summarizes the ****ordo** mappings of signifiers and signifieds.

0-9	decimal values
.	null token
=	barlines
==	double barline

Summary of ****ordo** *Signifiers*

EXAMPLES

A sample document is given below:

```
**ordo
=1
1
2
3
=2
4
.
5
*-
```

PERTINENT COMMANDS

The following Humdrum command produces **ordo data as output:

num	number selected Humdrum data records

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **ordo:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for **ordo

SEE ALSO

barlines (2), ****date** (2), ****dur** (2), ****metpos** (2), ****recip** (2), **scramble** (4), ****takt** (2), ****time** (2), ****Zeit** (2)

REPRESENTATION

****pc** — pitch-class representation

DESCRIPTION

The ****pc** representation can be used to characterize *pitch-class* or *chroma* information. Two pitches are said to share the same pitch-class or chroma when they are octave equivalents. No distinction is made between enharmonic spellings. Hence C-sharp4 and D-flat7 belong to the same pitch-class. In traditional set theory, pitch-classes are identified by integer values between 0 and 11, where C=0, C-sharp/D-flat=1 ... B=11.

Three types of data tokens are distinguished by ****pc**: pitch-class tokens, rest tokens, and barlines.

Pitch-class tokens are encoded as either numeric, or mixed alphabetic and numeric values. Following traditional practice, ****pc** encodes pitch-classes using the integer values between 0 and 11, where C=0, C-sharp/D-flat=1 ... B=11. The ****pc** representation provides aliases for the integers 10 and 11 ('A' and 'B' respectively). (Encodings using purely numeric values are especially useful when the representation is to be processed numerically. Encodings using mixed alphanumeric values are often better suited to pattern matching and searching tasks.†)

In addition to the basic pitch-class information, ****pc** also provides limited capabilities for representing articulation marks, ties, slurs, and phrasing. Slurs and phrase markings can be nested (slurs within slurs) and may also be elided to a single depth. Nested markings mean that one slur or phrase is entirely subsumed within another slur or phrase. For example: (( )) means that a short slur has occurred within a longer slur. Elisions are overlaps, for example, where an existing phrase fails to end before a new phrase begins. In ****pc** the ampersand character is used to mark elided slurs or phrases. For example: { &{ } &} means that a phrase begins, but does not end until after another phrase has begun. The ampersand is used to mark a matched pair of slur or phrases marks.

Rests are represented by the single letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

† For example, where '10' and '11' are present in a stream of data, the regular expression to search for pitch-class '1' is `(^|[^1])1([^01]|$)`, whereas the corresponding regular expression for alphanumeric pitch-class data is simply `1`.

FILE TYPE

It is recommended that files containing predominantly `**pc` data should be given names with the distinguishing '.pc' extension.

SIGNIFIERS

The following table summarizes the **pc mappings of signifiers and signifieds.

0-9	pitch-class values specified as integers or as real values; measure numbers
A	alias for pitch-class 10
B	alias for pitch-class 11
E	alias for pitch-class 11
r	rest
I	generic articulation (unspecified articulation)
O	generic ornament (unspecified ornament)
T	alias for pitch-class 10
=	barline; == double barline
[	first note of a tie
]	last note of a tie
_	middle note(s) of a tie (underscore)
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign
'	staccato mark
"	pizzicato mark
`	attacka mark
~	tenuto mark
^	accent mark
&	elision marker (for slurs or phases)

Summary of **pc *Signifiers*

EXAMPLES

The following sample document shows a pitch-class representation for the opening measures of Schoenberg's "Sommermüd" from *Three Songs,* opus 48. The left-most spine shows a mixed alphabetic and numeric encoding — where pitch-class 11 is represented by the letter 'B'.

```
!! Arnold Schoenberg, "Sommermued" (1933)
**pc        **pc        **pc       **text
*Ipiano     *Ipiano     *Ivox      *Deutsch
*M4/4       *M4/4       *M4/4      *M4/4
*MM72       *MM72       *MM72      *MM72
=1          =1          =1         =1
r           r           r          .
(7          (11         .          .
.           9^)         r          .
8')         .           1          Wenn
r           .           .          .
r           r           .          .
(9          (11         2          du
.           7^)         .          .
8')         .           0          schon
r           .           .          .
=2          =2          =2         =2
r           r           (0         glaubst ,
9           (7          [6         |
(9`         8           .          .
B)          .           .          .
r           .           .          .
r           7')         6])        |
.           r           .          .
.           (9          3          es
[B          .           3          ist
.           8)          [5         e-
=3          =3          =3         =3
*-          *-          *-         *-
```

Note that pitch-class representations do not preserve pitch-height or contour information.

PERTINENT COMMANDS

The following Humdrum commands accept ****pc** encoded data as inputs:

iv	determine interval vectors for vertical sonorities
nf	determine normal form for vertical sonorities
pc	convert between numerical and alphanumerical forms of ****pc**
pcset	convert to set-theoretic representations
pf	determine prime form for vertical sonorities
reihe	output tone-row variant for a given prime row
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces **pc data as output:

pc	translates ****cents**, ****freq**, ****kern**, ****pc**, ****pitch**, ****Tonh**, ****semits**, ****solfg**, ****specC**, to ****pc**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****pc**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for ****pc**

SEE ALSO

barlines (2), ****cents** (2), ****freq** (2), ****iv** (2), **iv** (4), ****kern** (2), ****nf** (2), **nf** (4), **pc** (4), ****pcset** (2), **pcset** (4), ****pf** (2), **pf** (4), ****pitch** (2), **reihe** (4), ****semits** (2), ****solfg** (2), ****specC** (2), ****Tonh** (2)

REFERENCES

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

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

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

REPRESENTATION

****pcset** — Fortean pitch-class set representation

DESCRIPTION

The ***pcset* representation provides a means for indicating pitch-class sets. The representation is based on a simple extension of the system devised by Allen Forte (see REFERENCE). Fortean set names consist of two numbers separated by a dash, with an optional up-case letter 'Z' preceding the second number. The first number indicates the pitch-class *cardinality* — that is, the number of unique pitch-classes in the set. For example, all major and minor chords have a cardinality of '3' since all such chords consist of three unique pitch-classes. (Inversionally-related pitch-class sets have the same set designation.)

The number following the dash simply distinguishes different pitch-class sets having the same cardinality. The letter 'Z' is used to indicate that the set shares the same interval-class content as some other set. (said to be Z-related sets). For example, sets 4-Z15 and 4-Z29 are said to be Z-related since they both exhibit the same interval-class content. (See `**iv` representation.)

Barlines are represented using the "common system" barlines described in Section 2 of this *Reference Manual.*

FILE TYPE

It is recommended that files containing predominantly `**pcs` data should be given names with the distinguishing '.pcs' extension.

SIGNIFIERS

The following table summarizes the ***pcset* mappings of signifiers and signifieds.

0-9	pitch-class set labels; measure numbers
.	null token
–	dash; separates cardinality for set number
=	barline; == double barline

*Summary of **pcset Signifiers*

EXAMPLES

The following sample document shows a pitch-class representation for the opening measures of Schoenberg's "Sommermüd" from *Three Songs,* Opus 48. The right-most spine shows a `**pcset` representation identifying the pitch-class content of each sonority. (The `**pcset` spine might be generated using the **pcset** command, after the `**pc` pitch-class spines are *filled-out* using the **fill** command.)

```
!! Arnold Schoenberg, "Sommermued" (1933)
**pc      **pc      **pc      **text    **pcset
*Ipiano   *Ipiano   *Ivoice   *Deutsch  *
*M4/4     *M4/4     *M4/4     *M4/4     *
*MM72     *MM72     *MM72     *MM72     *
=1        =1        =1        =1        =1
r         r         r         .         .
(7        (11       .         .         2-4
8')       .         1         Wenn      3-4
r         .         .         .         2-4
r         r         .         .         1-1
(9        (11       2         du        3-7
8')       .         0         schon     3-4
r         .         .         .         2-5
=2        =2        =2        =2        =2
r         r         (0        glaubst , 1-1
9         (7        [6        |         3-2
(9'       8         .         .         3-2
11)       .         .         .         3-7
r         .         .         .         2-2
r         7')       6])       |         2-1
[11       .         3         ist       3-8
=3        =3        =3        =3        =3
*-        *-        *-        *-        *-
```

PERTINENT COMMANDS

The following humdrum commands accept ***pcset* encoded data as inputs:

iv	determine interval vectors for successive vertical sonorities
nf	determine normal form for successive vertical sonorities
pcset	convert pitch and pitch-class information to set-theoretic representations
pf	prime form representation

The following Humdrum command produces ***pcset* data as output:

pcset	convert pitch and pitch-class information to set-theoretic representations

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ***pcset:*

meter signatures	*M6/8
key signatures	*k[ ]
key	*c#:
tempo	*MM96.3
timebase	*tb32

*Tandem interpretations for **pcset*

SEE ALSO

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

REFERENCES

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

APPENDIX

The following table provides an extended list of all possible Fortean-type pitch-class set names. The corresponding pitch-class set and interval vectors are also shown.

Fortean set-name	pitch-class set	interval vector	Fortean set-name	pitch-class set	interval vector
0-0	(empty)	<000000>	12-1	(0,1,2,3,4,5,6,7,8,9,10,11)	<12,12,12,12,12,12>
1-1	(0)	<000000>	11-1	(0,1,2,3,4,5,6,7,8,9,10)	<10,10,10,10,10,10>
2-1	(0,1)	<100000>	10-1	(0,1,2,3,4,5,6,7,8,9)	<988888>
2-2	(0,2)	<010000>	10-2	(0,1,2,3,4,5,6,7,8,A)	<898888>
2-3	(0,3)	<001000>	10-3	(0,1,2,3,4,5,6,7,9,A)	<889888>
2-4	(0,4)	<000100>	10-4	(0,1,2,3,4,5,6,8,9,A)	<888988>
2-5	(0,5)	<000010>	10-5	(0,1,2,3,4,5,7,8,9,A)	<888898>
2-6	(0,6)	<000001>	10-6	(0,1,2,3,4,6,7,8,9,A)	<888889>
3-1	(0,1,2)	<210000>	9-1	(0,1,2,3,4,5,6,7,8)	<876663>
3-2	(0,1,3)	<111000>	9-2	(0,1,2,3,4,5,6,7,9)	<777663>
3-3	(0,1,4)	<101100>	9-3	(0,1,2,3,4,5,6,8,9)	<767763>
3-4	(0,1,5)	<100110>	9-4	(0,1,2,3,4,5,7,8,9)	<766773>
3-5	(0,1,6)	<100011>	9-5	(0,1,2,3,4,6,7,8,9)	<766674>
3-6	(0,2,4)	<020100>	9-6	(0,1,2,3,4,5,6,8,10)	<686763>
3-7	(0,2,5)	<011010>	9-7	(0,1,2,3,4,5,7,8,10)	<677673>
3-8	(0,2,6)	<010101>	9-8	(0,1,2,3,4,6,7,8,10)	<676764>
3-9	(0,2,7)	<010020>	9-9	(0,1,2,3,5,6,7,8,10)	<676683>
3-10	(0,3,6)	<002001>	9-10	(0,1,2,3,4,6,7,9,10)	<668664>
3-11	(0,3,7)	<001110>	9-11	(0,1,2,3,5,6,7,9,10)	<667773>
3-12	(0,4,8)	<000300>	9-12	(0,1,2,4,5,6,8,9,10)	<666963>
4-1	(0,1,2,3)	<321000>	8-1	(0,1,2,3,4,5,6,7)	<765442>
4-2	(0,1,2,4)	<221100>	8-2	(0,1,2,3,4,5,6,8)	<665542>
4-3	(0,1,3,4)	<212100>	8-3	(0,1,2,3,4,5,6,9)	<656542>
4-4	(0,1,2,5)	<211110>	8-4	(0,1,2,3,4,5,7,8)	<655552>
4-5	(0,1,2,6)	<210111>	8-5	(0,1,2,3,4,6,7,8)	<654553>
4-6	(0,1,2,7)	<210021>	8-6	(0,1,2,3,5,6,7,8)	<654463>
4-7	(0,1,4,5)	<201210>	8-7	(0,1,2,3,4,5,8,9)	<645652>

4-8	(0,1,5,6)	<200121>	8-8	(0,1,2,3,4,7,8,9)	<644563>
4-9	(0,1,6,7)	<200022>	8-9	(0,1,2,3,6,7,8,9)	<644464>
4-10	(0,2,3,5)	<122010>	8-10	(0,2,3,4,5,6,7,9)	<566452>
4-11	(0,1,3,5)	<121110>	8-11	(0,1,2,3,4,5,7,9)	<565552>
4-12	(0,2,3,6)	<112101>	8-12	(0,1,3,4,5,6,7,9)	<556543>
4-13	(0,1,3,6)	<112011>	8-13	(0,1,2,3,4,6,7,9)	<556453>
4-14	(0,2,3,7)	<111120>	8-14	(0,1,2,4,5,6,7,9)	<555562>
4-Z15	(0,1,4,6)	<111111>	8-Z15	(0,1,2,3,4,6,8,9)	<555553>
4-16	(0,1,5,7)	<110121>	8-16	(0,1,2,3,5,7,8,9)	<554563>
4-17	(0,3,4,7)	<102210>	8-17	(0,1,3,4,5,6,8,9)	<546652>
4-18	(0,1,4,7)	<102111>	8-18	(0,1,2,3,5,6,8,9)	<546553>
4-19	(0,1,4,8)	<101310>	8-19	(0,1,2,4,5,6,8,9)	<545752>
4-20	(0,1,5,8)	<101220>	8-20	(0,1,2,4,5,7,8,9)	<545662>
4-21	(0,2,4,6)	<030201>	8-21	(0,1,2,3,4,6,8,10)	<474643>
4-22	(0,2,4,7)	<021120>	8-22	(0,1,2,3,5,6,8,10)	<465562>
4-23	(0,2,5,7)	<021030>	8-23	(0,1,2,3,5,7,8,10)	<465472>
4-24	(0,2,4,8)	<020301>	8-24	(0,1,2,4,5,6,8,10)	<464743>
4-25	(0,2,6,8)	<020202>	8-25	(0,1,2,4,6,7,8,10)	<464644>
4-26	(0,3,5,8)	<012120>	8-26	(0,1,2,4,5,7,9,10)	<456562>
4-27	(0,2,5,8)	<012111>	8-27	(0,1,2,4,5,7,8,10)	<456553>
4-28	(0,3,6,9)	<004002>	8-28	(0,1,3,4,6,7,9,10)	<448444>
4-Z29	(0,1,3,7)	<111111>	8-Z29	(0,1,2,3,5,6,7,9)	<555553>
5-1	(0,1,2,3,4)	<432100>	7-1	(0,1,2,3,4,5,6)	<654321>
5-2	(0,1,2,3,5)	<332110>	7-2	(0,1,2,3,4,5,7)	<554331>
5-3	(0,1,2,4,5)	<322210>	7-3	(0,1,2,3,4,5,8)	<544431>
5-4	(0,1,2,3,6)	<322111>	7-4	(0,1,2,3,4,6,7)	<544332>
5-5	(0,1,2,3,7)	<321121>	7-5	(0,1,2,3,5,6,7)	<543342>
5-6	(0,1,2,5,6)	<311221>	7-6	(0,1,2,3,4,7,8)	<533442>
5-7	(0,1,2,6,7)	<310132>	7-7	(0,1,2,3,6,7,8)	<532353>
5-8	(0,2,3,4,6)	<232201>	7-8	(0,2,3,4,5,6,8)	<454422>
5-9	(0,1,2,4,6)	<231211>	7-9	(0,1,2,3,4,6,8)	<453432>
5-10	(0,1,3,4,6)	<223111>	7-10	(0,1,2,3,4,6,9)	<445332>
5-11	(0,2,3,4,7)	<222220>	7-11	(0,1,3,4,5,6,8)	<444441>
5-Z12	(0,1,3,5,6)	<222121>	7-Z12	(0,1,2,3,4,7,9)	<444342>
5-13	(0,1,2,4,8)	<221311>	7-13	(0,1,2,4,5,6,8)	<443532>
5-14	(0,1,2,5,7)	<221131>	7-14	(0,1,2,3,5,7,8)	<443352>
5-15	(0,1,2,6,8)	<220222>	7-15	(0,1,2,4,6,7,8)	<442443>
5-16	(0,1,3,4,7)	<213211>	7-16	(0,1,2,3,5,6,9)	<435432>
5-Z17	(0,1,3,4,8)	<212320>	7-Z17	(0,1,2,4,5,6,9)	<434541>
5-Z18	(0,1,4,5,7)	<212221>	7-Z18	(0,1,2,3,5,8,9)	<434442>
5-19	(0,1,3,6,7)	<212122>	7-19	(0,1,2,3,6,7,9)	<434343>
5-20	(0,1,3,7,8)	<211231>	7-20	(0,1,2,4,7,8,9)	<433452>
5-21	(0,1,4,5,8)	<202420>	7-21	(0,1,2,4,5,8,9)	<424641>
5-22	(0,1,4,7,8)	<202321>	7-22	(0,1,2,5,6,8,9)	<424542>
5-23	(0,2,3,5,7)	<132130>	7-23	(0,2,3,4,5,7,9)	<354351>
5-24	(0,1,3,5,7)	<131221>	7-24	(0,1,2,3,5,7,9)	<353442>
5-25	(0,2,3,5,8)	<123121>	7-25	(0,2,3,4,6,7,9)	<345342>
5-26	(0,2,4,5,8)	<122311>	7-26	(0,1,3,4,5,7,9)	<344532>
5-27	(0,1,3,5,8)	<122230>	7-27	(0,1,2,4,5,7,9)	<344451>
5-28	(0,2,3,6,8)	<122212>	7-28	(0,1,3,5,6,7,9)	<344433>
5-29	(0,1,3,6,8)	<122131>	7-29	(0,1,2,4,6,7,9)	<344352>
5-30	(0,1,4,6,8)	<121321>	7-30	(0,1,2,4,6,8,9)	<343542>
5-31	(0,1,3,6,9)	<114112>	7-31	(0,1,3,4,6,7,9)	<336333>
5-32	(0,1,4,6,9)	<113221>	7-32	(0,1,3,4,6,8,9)	<335442>
5-33	(0,2,4,6,8)	<040402>	7-33	(0,1,2,4,6,8,10)	<262623>
5-34	(0,2,4,6,9)	<032221>	7-34	(0,1,3,4,6,8,10)	<254442>

5-35	(0,2,4,7,9)	<032140>	7-35	(0,1,3,5,6,8,10)	<254361>
5-Z36	(0,1,2,4,7)	<222121>	7-Z36	(0,1,2,3,5,6,8)	<444342>
5-Z37	(0,3,4,5,8)	<212320>	7-Z37	(0,1,3,4,5,7,8)	<434541>
5-Z38	(0,1,2,5,8)	<212221>	7-Z38	(0,1,2,4,5,7,8)	<434442>
6-1	(0,1,2,3,4,5)	<543210>	6-Z26	(0,1,3,5,7,8)	<232341>
6-2	(0,1,2,3,4,6)	<443211>	6-27	(0,1,3,4,6,9)	<225222>
6-Z3	(0,1,2,3,5,6)	<433221>	6-Z28	(0,1,3,5,6,9)	<224322>
6-Z4	(0,1,2,4,5,6)	<432321>	6-Z29	(0,1,3,6,8,9)	<224232>
6-5	(0,1,2,3,6,7)	<422232>	6-30	(0,1,3,6,7,9)	<224223>
6-Z6	(0,1,2,5,6,7)	<421242>	6-31	(0,1,3,5,8,9)	<223431>
6-7	(0,1,2,6,7,8)	<420243>	6-32	(0,2,4,5,7,9)	<143250>
6-8	(0,2,3,4,5,7)	<343230>	6-33	(0,2,3,5,7,9)	<143241>
6-9	(0,1,2,3,5,7)	<342231>	6-34	(0,1,3,5,7,9)	<142422>
6-Z10	(0,1,3,4,5,7)	<333321>	6-35	(0,2,4,6,8,10)	<060603>
6-Z11	(0,1,2,4,5,7)	<333231>	6-Z36	(0,1,2,3,4,7)	<433221>
6-Z12	(0,1,2,4,6,7)	<332232>	6-Z37	(0,1,2,3,4,8)	<432321>
6-Z13	(0,1,3,4,6,7)	<324222>	6-Z38	(0,1,2,3,7,8)	<421242>
6-14	(0,1,3,4,5,8)	<323430>	6-Z39	(0,2,3,4,5,8)	<333321>
6-15	(0,1,2,4,5,8)	<323421>	6-Z40	(0,1,2,3,5,8)	<333231>
6-16	(0,1,4,5,6,8)	<322431>	6-Z41	(0,1,2,3,6,8)	<332232>
6-Z17	(0,1,2,4,7,8)	<322332>	6-Z42	(0,1,2,3,6,9)	<324222>
6-18	(0,1,2,5,7,8)	<322242>	6-Z43	(0,1,2,5,6,8)	<322332>
6-Z19	(0,1,3,4,7,8)	<313431>	6-Z44	(0,1,2,5,6,9)	<313431>
6-20	(0,1,4,5,8,9)	<303630>	6-Z45	(0,2,3,4,6,9)	<234222>
6-21	(0,2,3,4,6,8)	<242412>	6-Z46	(0,1,2,4,6,9)	<233331>
6-22	(0,1,2,4,6,8)	<241422>	6-Z47	(0,1,2,4,7,9)	<233241>
6-Z23	(0,2,3,5,6,8)	<234222>	6-Z48	(0,1,2,5,7,9)	<232341>
6-Z24	(0,1,3,4,6,8)	<233331>	6-Z49	(0,1,3,4,7,9)	<224322>
6-Z25	(0,1,3,5,6,8)	<233241>	6-Z50	(0,1,4,6,7,9)	<224232>

REPRESENTATION

****pitch** — American National Standards Institute pitch notation

DESCRIPTION

The ****pitch** representation permits the encoding of Western musical pitches using the standard system for pitch designations adopted by the American National Standards Institute (e.g. 'G#4').

The ****pitch** representation distinguishes three types of data tokens: pitches, rests, and barlines. Pitches may be encoded using pitch letter-name, accidental, octave designation, and pitch deviation (in cents). In addition, ****pitch** provides limited capabilities for representing phrasing and slurs.

Pitch tokens consist of up to four logical parts — without any intervening spaces. The first part is the pitch letter-name; only the upper-case letters A, B, C, D, E, F, and G are permitted. The second part is an optional accidental: the octothorpe (#) for sharp, the small letter 'b' for flat, the small letter 'x' for double-sharp, and two successive small letter 'b's for double-flats. Triple and quadruple sharps (x#, xx, etc.) and flats (bbb, bbbb, etc.) are permissible, however a given pitch token may not be modified by sharps and flats concurrently.

The third part of a pitch token is the octave designation. The number 4 is used to designate all pitches between middle C and the B a major seventh above, inclusive. Octave numbers are incremented by one for each successively higher octave, and are decremented by one for each successively lower octave. Negative octave numbers are not permitted, so the lowest pitch in the ****pitch** representation is C0 (16.35 Hz). Only a single octave digit is permitted, so the highest ****pitch** pitch is B9 (15,804 Hz).

If a given pitch deviates from equal temperament, a fourth part is added to a ****pitch** pitch token denoting the cents deviation from equal temperament. This consists of an integer value preceded by either a plus or minus sign.

By way of example, the following ****pitch** token encodes a note 19 cents flat from A440 (in this case A435 Hz):

```
A4-19
```

Once again, no intervening spaces are permitted within a single note. Notice that the order of signifiers is important for pitch encodings. Pitch letter-name is followed by one or more accidentals (if appropriate), followed by an octave designation, followed by cents deviation (if appropriate) — where cents deviation is an integer preceded by the plus or minus sign.

Several notes may be encoded concurrently in a single spine by using the Humdrum multiple-stop convention: notes within multiple-stops are separated by single spaces. The following example encodes a C-minor chord as four pitches in two **pitch** spines — each spine containing a double-stop.

```
**pitch **pitch
C4 Eb4  G4 C5
*-      *-
```

Pitch tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The open parenthesis '(' denotes the beginning of a slur. The closed parenthesis ')' denotes the end of a slur. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**pit` data should be given names with the distinguishing '.pit' extension.

SIGNIFIERS

The following table summarizes the ****pitch** mappings of signifiers and signifieds.

A-G	absolute pitches letter name
0-9	octave designation, where C4 equals middle C; also cents deviation (when preceded by minus or plus sign)
b	flat
bb	double flat
bbb	triple flat
#	sharp
x	double sharp
x#	triple sharp
xx	quadruple sharp
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign
–	negative cents deviation (minus sign)
+	positive cents deviation

Summary of ****pitch** *Signifiers*

EXAMPLES

A sample document is given below:

```
**pitch   **pitch
*pelog    *slendro
=1        =1
A#4+20    Bb4-80
A4        .
A4        .
=2        =2
A4        {(G3+5
r         G3+73
.         G#3-2)
r         r
r         Db4+18
Db4       Db4+77}
*-        *-
```

PERTINENT COMMANDS

The following Humdrum commands accept **pitch** encoded data as inputs:

cents	translates **pitch** to **cents**
deg	translate **pitch** to **deg**
degree	translate **pitch** to **degree**
freq	translates **pitch** to **freq**
hint	calculate harmonic intervals from **pitch** input
kern	translates **pitch** to **kern**
mint	calculate melodic intervals from **pitch** input
pc	translate **pitch** pitch to **pc**
semits	translate **pitch** pitch to numerical **semits**
solfa	translate **pitch** pitch to **solfa**
solfg	translate **pitch** pitch to **solfg**
tonh	translate **pitch** pitch to **Tonh**
trans	transpose **pitch** score
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces **pitch** data as output:

pitch	translates **cents**, **degree**, **freq**, **fret**, **kern**, **MIDI**, **semits**, **solfg**, **specC**, and **Tonh** to **pitch**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **pitch**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`

Tandem interpretations for **pitch**

SEE ALSO

barlines (2), **cents** (2), **deg** (2), **degree** (2), **freq** (2), **hint** (2), **kern** (2), **mint** (4), **pc** (2), **pitch** (4), **semits** (2), **solfa** (2), **solfg** (2), **specC** (2), **Tonh** (2)

REPRESENTATION

**recip — beat-proportion representation

DESCRIPTION

The **recip scheme is able to represent durations according to the traditional system of beat-proportions (rather than according to elapsed- or clock-time.) Durations are specified through the use of integer numbers and augmentation dots. With the exception of the value zero, durations are represented by reciprocal numerical values. For example:

0	breve duration
1	whole duration
2	half duration
4	quarter duration
8	eighth duration
16	sixteenth duration
32	thirty-second duration
64	sixty-fourth duration
etc.	

Representations of Duration in **recip

The number zero (0) is reserved for the *breve* duration (i.e. a duration of twice the length of a whole note). Dotted durations are indicated by adding the period character (.) after the numerical value:

2.	dotted half duration
8..	doubly-dotted eighth duration
etc.	

Representation of Dotted Durations in **recip

Any number of augmentation dots may follow the duration integer.†

Triplet and other irregular durations are represented in a somewhat more arcane, though no less logical fashion. Consider, for example, the quarter-note triplet duration. Three quarter triplets occur in the time of four quarters or one whole duration. If we divide a whole duration ("1") into three equal parts, each part has a duration of one-third. The corresponding reciprocal integer for 1/3 is 3, hence **recip represents a quarter-note triplet

† Notice that the period is used both to indicate Humdrum null tokens and **recip augmentation dots. In parsing **recip spines, there is never any confusion concerning the meaning of the period: as a null token, the period will appear isolated from all other characters (by tabs or carriage returns). As an augmentation dot, the period will always follow a number.

as a "third-note" — 3. Similarly, eighth-note triplets are represented by the integer 6 while sixteenth-note triplets are represented by the integer 12. Eighth-note quintuplets (5 in the time of 4) will be represented by the value 10 (a half duration divided by 5).

In general, the way to determine the ****recip** equivalent of an arbitrary "tuplet" duration is to multiply the number of tuplets by the total duration which they occupy. If 7 notes of equal duration occupy the duration of a whole-note ("1"), then each septuplet is represented by the value 7 (i.e. 1 x 7). A more extreme example is 23 notes in the time of a doubly-dotted quarter. The appropriate ****recip** duration can be found by multiplying 4 by 23 (equals 92) and adding the appropriate augmentation dots. Thus "92.." is the correct ****recip** encoding for a note whose duration is 23 notes in the time of a doubly-dotted quarter.

The ****recip** representation can be used to encode a sequence of time-spans or successive durations. The units are seconds.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**recip` data should be given names with the distinguishing '.rcp' extension.

SIGNIFIERS

The following table summarizes the ****recip** mappings of signifiers and signifieds.

0-9	decimal values
.	dotted duration; null token
=	barlines
==	double barline

Summary of ****recip** *Signifiers*

EXAMPLES

A sample document is given below:

```
!! Gustav Holst
**recip
*M5/4
=1
12
12
12
4
4
8
8
4
=2
*-
```

PERTINENT COMMANDS

The following Humdrum command accepts ****recip** encoded data as input:

timebase	reformat ****recip** score with constant time-base
urrhythm	characterize the rhythmic prototype in a passage

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****recip**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for ****recip**

SEE ALSO

barlines (2), ****date** (2), ****metpos** (2), ****ordo** (2), ****recip** (2), ****takt** (2), ****time** (2), **timebase** (4), ****URrhythm** (2), **urrhythm** (4), ****Zeit** (2)

REPRESENTATION

****semits** — semitone absolute pitch representation

DESCRIPTION

The ****semits** representation is used to represent absolute pitch in semitone units with respect to middle C. Middle C is designated zero semitones. All other pitches are represented with respect to this reference, hence A4 is 9 semits and A3 is -3 semits. Fractional values can also be represented, hence 1.5 is midway between C#4 and D4.

Pitch tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The open parenthesis '(' denotes the beginning of a slur. The closed parenthesis ')' denotes the end of a slur. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2).**

FILE TYPE

It is recommended that files containing predominantly `**semits` data should be given names with the distinguishing '.sem' extension.

SIGNIFIERS

The following table summarizes the ****semits** mappings of signifiers and signifieds.

0-9	decimal values
–	minus sign
+	plus sign (optional)
.	fractional delimiter; null token
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign

Summary of ****semits** *Signifiers*

EXAMPLES

The following sample document shows a `**kern` spine with corresponding `**semits` pitch values.

```
**semits    **kern
*M4/4       *M4/4
*c:         *c:
=           =
r           8r
12          16cc
11          16bn
12          8cc
14          8dd
5           16f
7           16g
8           4a-
7           16g
5           16f
=           =
4 7         1e 1g
==          ==
*-          *-
```

PERTINENT COMMANDS

The following Humdrum commands accept ****semits** encoded data as inputs:

cents	translates ****semits** to ****cents**
freq	translates ****semits** to ****freq**
kern	translates ****semits** to ****kern**
melac	calculate melodic accent values for successive pitches
pc	translate ****semits** pitch to ****pc**
pitch	translate ****semits** pitch to ****pitch**
semits	change numerical precision of ****semits** values
solfg	translate ****semits** pitch to ****solfg**
tonh	translate ****semits** pitch to ****Tonh**
vox	determine active and inactive voices in a Humdrum file

The following Humdrum commands produce ****semits** data as outputs:

semits	translates ****cents**, ****freq**, ****fret**, ****kern**, ****MIDI**, ****pitch**, ****semits**, ****solfg**, ****specC**, ****Tonh**, to ****semits**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****semits:**

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****semits**

SEE ALSO

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

REPRESENTATION

****simil** — similarity representation

DESCRIPTION

The ****simil** representation encodes numerical values that indicate the Damerau-Levenshtein edit distance between two other Humdrum representations. Data tokens for ****simil** consist only of decimal numbers. The value 1.0 indicates identical material beginning at the current data record. Lower numerical values indicate increasingly less similarity at the current data record.

FILE TYPE

It is recommended that files containing predominantly `**simil` data should be given names with the distinguishing '.sim' extension.

SIGNIFIERS

The following table summarizes the ****simil** mappings of signifiers and signifieds.

0-9	decimal values
.	fractional delimiter; null token

Summary of ****simil** *Signifiers*

EXAMPLES

A sample document is given below:

```
**simil
0.78
0.73
0.74
0.69
0.71
0.68
0.68
0.63
.
.
*-
```

PERTINENT COMMANDS

The following Humdrum command produces ****simil** data as output:

simil measure the similarity between two Humdrum spines

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****simil**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****simil**

SEE ALSO

cmp (UNIX), ****correl** (2), **correl** (4), **diff** (UNIX), **patt** (4), **pattern** (4), **simil** (4), ****xref** (2)

REPRESENTATION

**solfa — representation for tonic solfa syllables

DESCRIPTION

The **solfa representation permits the encoding of extended tonic solfa syllables for the "moveable doh" system of pitch naming.

The **solfa representation distinguishes three types of data tokens: pitches, rests, and barlines.

Pitches tokens are represented using the syllables *do, re, mi, fa, so, la,* and *ti* — or their chromatic alterations: *di, da, ri, ra,* etc. (see table 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 or C minor.

The **solfa representation does not distinguish between major and minor modes. Only the tonic pitch is of importance when determining the representation for a given pitch. For example, in both C major and C minor, the pitch A-natural is represented as `la` while the pitch A-flat is represented as `le`.

The amount of chromatic alteration is not represented by **solfa; once a pitch is "raised," raising it further will not change the note's representation. For example, where the tonic pitch is B-flat, both B-natural and B-sharp are represented by `di`.

Octave designations are not represented in **solfa. However, **solfa provides limited capabilities for representing phrasing and slurs.

Several pitches may be encoded concurrently in a single spine by using the Humdrum multiple-stop convention: pitches within multiple-stops are separated by single spaces. For example, the following example encodes a 4-note tonic major chord using two **solfa spines — each spine containing a double-stop.

```
**solfa   **solfa
*C:       *C:
do mi     so do
*-        *-
```

Pitch tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The open parenthesis '(' denotes the beginning of a slur. The closed parenthesis ')' denotes the end of a slur. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

File type is dubbed '.sol'.

SIGNIFIERS

The following table summarizes the ****solfa** mappings of signifiers and signifieds.

do	(pronounced *doe*) tonic pitch
di	(*dee*) raised tonic pitch
de	(*day*) lowered tonic pitch
re	(*ray*) supertonic pitch
ri	(*ree*) raised supertonic pitch
ra	(*raw*) lowered supertonic pitch
mi	(*me*) mediant pitch
my	(*my*) raised mediant pitch
me	(*may*) lowered mediant pitch
fa	(*fah*) subdominant pitch
fi	(*fee*) raised subdominant pitch
fe	(*fay*) lowered subdominant pitch
so	(*so*) dominant pitch
si	(*see*) raised dominant pitch
se	(*say*) lowered dominant pitch
la	(*la*) submediant pitch
li	(*lee*) raised submediant pitch
le	(*lay*) lowered submediant pitch
ti	(*tee*) leading tone
ty	(*tie*) raised leading tone
te	(*tay*) lowered leading tone
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign

Summary of ****solfa** *Signifiers*

EXAMPLES

A sample document is given below:

```
!! Johannes Brahms
!! Waltz Opus 39, No. 15
**solfa         **solfa
*M3/4           *M3/4
*A:             *A:
=2              =2
do              so mi
so mi           .
.               mi do
mi so           mi do
.               so mi
=3              =3
do              la fa
.               so
.               fa
fa fa           so mi
la do           fa re
=4              =4
do              so mi
*-              *-
```

PERTINENT COMMANDS

The following Humdrum command accepts ****solfa** encoded data as inputs:

vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces ****solfa** data as output:

solfa	translates ****kern**, ****pitch**, ****solfg**, ****Tonh**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****solfa**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`

Tandem interpretations for ****solfa**

SEE ALSO

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

REPRESENTATION

****solfg** — French solfège (pitch) notation

DESCRIPTION

The ****solfg** representation permits the encoding of Western musical pitches using the common French system for pitch naming.

The ****solfg** representation distinguishes three types of data tokens: pitches, rests, and barlines. Pitches are encoded using diatonic pitch names, accidentals, and octave indications. In addition, ****solfg** provides limited capabilities for representing phrasing and slurs.

Pitch tokens consist of three logical parts — without any intervening spaces. The first part uses the so-called 'fixed-do' method of diatonic pitch designations: *do, ré, mi, fa, sol, la,* and *si.* The second part is an optional accidental preceded by an tilde. Flats (*bémol*) and sharps (*diese*) are abbreviated *b* and *d* respectively. Hence, 'do diese' (`do~d`) for C-sharp, 'la beamol' (`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 third part of a pitch token is the octave designation. The number 4 is used to designate all pitches between middle C and the *si* a major seventh above, inclusive. Octave numbers are incremented by one for each successively higher octave, and are decremented by one for each successively lower octave. Negative octave numbers are not permitted, so the lowest pitch in the ****solfg** representation is do0 (16.35 Hz). Only a single octave digit is permitted, so the highest ****solfg** pitch is si9 (15,804 Hz).

Once again, no intervening spaces are permitted within a single note. Notice that the order of signifiers is important for pitch encodings. Pitch letter-name is followed by one or more accidentals (if appropriate), followed by an octave designation.

Several notes may be encoded concurrently in a single spine by using the Humdrum multiple-stop convention: notes within multiple-stops are separated by single spaces. The following example encodes a C-minor chord as four pitches in two ****solfg** spines — each spine containing a double-stop.

```
**solfg      **solfg
do4 mi~b4    sol4 do5
*-           *-
```

Pitch tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The open parenthesis '(' denotes the beginning of a slur. The closed parenthesis ')' denotes the end of a slur. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**solfg` data should be given names with the distinguishing '.slg' extension.

SIGNIFIERS

The following table summarizes the ****solfg** mappings of signifiers and signifieds.

0-9	octave designation, where do4 equals middle C;
doremifasl	diatonic pitches letter names
~	accidental delimiter
~b	flat (bémol)
~d	sharp (dièse)
~bb	double-flat
~dd	double-sharp
n	natural
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign

Summary of ****solfg** *Signifiers*

EXAMPLES

A sample document is given below:

```
!! Claude Debussy
!! "Voiles"
**solfg        **solfg
=1             =1
r              r
{mi5           {sol~d5
re5            fah~d5
do5            mi5
si~b4          re5
=2             =2
la~b4          do5
sol~d5         do6
fa~d5}         si~b5}
=3             =3
*-             *-
```

PERTINENT COMMANDS

The following Humdrum commands accept ****solfg** encoded data as inputs:

cents	translate ****solfg** to ****cents**
deg	translate ****solfg** to ****deg**
degree	translate ****solfg** to ****degree**
freq	translate ****solfg** to ****freq**
hint	calculate harmonic intervals from ****solfg** input
kern	translate ****solfg** to ****kern**
mint	calculate melodic intervals from ****solfg** input
pc	translate ****solfg** pitch to ****pc**
pitch	translate ****solfg** pitch to ****pitch**
semits	translate ****solfg** pitch to numerical ****semits**
solfa	translate ****solfg** pitch to ****solfa**
tonh	translate ****solfg** pitch to ****Tonh**
trans	transpose ****solfg** score
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces ****solfg** data as output:

solfg	translates ****cents**, ****degree**, ****fret**, ****freq**, ****kern**, ****MIDI**, ****pitch**, ****semits**, ****specC**, and ****Tonh** to ****solfg**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****solfg**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`

Tandem interpretations for ****solfg**

SEE ALSO

barlines (2), ****cents** (2), ****deg** (2), ****degree** (2), ****freq** (2), ****hint** (2), ****kern** (2), ****mint** (2), **mint** (4), ****pc** (2), **pc** (4), ****pitch** (2), **pitch** (4), ****semits** (2), ****solfa** (2), **solfg** (4), ****Tonh** (2), **tonh** (4)

REPRESENTATION

****specC** — spectral centroid representation

DESCRIPTION

The ****specC** representation can be used to represent the frequency of the spectral centroid for pure or complex tones. Spectral centroid is the amplitude-weighted mean of several frequency components. For example, given three frequencies (200, 400, 900 Hz) of equal amplitude, the corresponding spectral centroid would be the arithmetic mean of the three frequencies (i.e. 750 Hz). For pure tones, the spectral centroid is equivalent to the frequency of the pure tone. For complex tones, the spectral centroid is higher when tones have greater energy (amplitudes) in the upper partials. Hence, spectral centroid provides a simple index for the *richness* or *brightness* of a tone or sonority.

Spectral centroid tokens are numerical values in units of hertz. Frequencies may be specified as integer or real values (using a decimal).

Barlines are represented using the "common system" for barlines — see **barlines (2).**

FILE TYPE

It is recommended that files containing predominantly `**specC` data should be given names with the distinguishing '.spc' extension.

SIGNIFIERS

The following table summarizes the ****specC** mappings of signifiers and signifieds.

0-9	frequency (in hertz) specified as an integer or real value
r	rest
=	barline; == double barline

Summary of ****specC** *Signifiers*

EXAMPLES

A sample document is given below:

```
**specC   **specC
*pure     *complex
=1        =1
1900      2730
868.9     .
1362      4402
2263.     .
.         3742
=2        =2
r         r
==        ==
*-        *-
```

Notice that frequencies can be either real or integer values. Rests are represented by the single letter 'r'.

PERTINENT COMMANDS

The following Humdrum commands accept ****specC** encoded data as inputs:

cents	translates ****specC** to ****cents**
deg	translate ****specC** to ****deg**
degree	translate ****specC** to ****degree**
freq	translate ****specC** to ****freq**
kern	translates ****specC** to ****kern**
pc	translate ****specC** pitch to ****pc**
pitch	translates ****specC** to ****pitch**
semits	translate ****specC** to numerical ****semits**
solfg	translate ****specC** pitch to ****solfg**
specc	change numerical precision of ****specC** values
tonh	translate ****solfg** pitch to ****Tonh**

The following Humdrum command produces ****specC** data as output:

specc	translates ****freq**, and ****specC** to ****specC**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****specC**:

pure tones	`*pure`
complex tones	`*complex`
meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for ****specC**

SEE ALSO

barlines (2), ****cents** (2), ****deg** (2), ****degree** (2), ****freq** (2), ****kern** (2), ****pc** (2), ****pitch** (2), ****semits** (2), ****solfg** (2), ****Tonh** (2)

REPRESENTATION

****spect** — discrete frequency spectrum representation

DESCRIPTION

The ****spect** representation is used to represent successive acoustic spectra. Three types of data tokens are recognized by ****spect**: spectral data, silence (ambient spectrum), and barlines. In the case of spectral data, each token represents a complete spectrum specified as a set of concurrent discrete frequency components. Each component in the spectrum is represented by a pair of numerical values separated by a semicolon (;). These paired values encode the frequency and amplitude attributes respectively for a single spectral component. Frequency values are positive integer or real values representing *hertz*. Amplitude values are positive integer or real values representing the sound pressure level in decibels (dB SPL). Where a spectrum consists of more than one pure tone component, the data are encoded as Humdrum multiple-stops.

When no sound is present, this is represented by an ambient spectrum — denoted simply by the upper-case letter 'A'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**spect` data should be given names with the distinguishing '.spe' extension.

SIGNIFIERS

The following table summarizes the ****spect** mappings of signifiers and signifieds.

0-9	frequency or amplitude values specified as integer or real value; measure numbers
.	decimal point; null token
A	ambient spectrum ("silence")
=	barline; == double barline
=;	barline with pause sign

Summary of ****spect** *Signifiers*

EXAMPLES

The following sample document encodes five spectra and a barline. The first data record encodes an ambient spectrum (silence). There ensues two spectra, each consisting of three spectral components. The first spectrum consists of a 261 Hz tone at 47 dB SPL, as well as spectral components at 523 Hz and 785 Hz at 57 dB SPL and 35 dB SPL, respectively.

Following the barline are two data records that encode two different amalgamations of the preceding two three-component spectra. Notice that these two spectra are identical; only the order of the components differs. In the **spect representation there is no special requirement that the spectral components be encoded in any particular order.

```
**spect
A
261;47 523;57 785;35
330;57 659;35 989;27
=1
261;47 523;57 785;35 330;57 659;35 989;27
261;47 330;57 523;57 659;35 785;35 989;27
*-
```

PERTINENT COMMANDS

The following Humdrum command produces **spect data as output:

spect assemble total spectral content for individual sonorities

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **spect:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`

Tandem interpretations for **spect

SEE ALSO

barlines (2), **dB (2), **diss** (4), **freq (2), **semits (2), **semits** (4), **spect (2), **spect** (4)

WARNING

The **spect representation frequently produces long data records. Viewing or editing the output using restricted window widths may be inconvenient.

REPRESENTATION

**synco — represent degree of metric syncopation

DESCRIPTION

The **synco representation encodes numerical values that indicate the degree of metric syncopation for successive moments in a musical passage. Data tokens for **synco consist only of decimal numbers. The value 0 indicates no metric syncopation at the current data record. Higher numerical values indicate increasing amounts of metric syncopation at the current data record.

Syncopation values follow a definition of metric syncopation inspired by the work of Lee and Longuet-Higgins (1982). 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 occurs on the second beat of a quadruple meter, and is not followed by a note on the third beat, the third beat is deemed syncopated because it occupies a higher metric position than the previous onset.

Numerical **synco values are equal to 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. For example, missing downbeats at the beginning of a measure produce the large syncopation values.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

Note: **synco data is normally produced by the **synco** command.

FILE TYPE

It is recommended that files containing predominantly `**synco` data should be given names with the distinguishing '.syn' extension.

SIGNIFIERS

The following table summarizes the **synco mappings of signifiers and signifieds.

0-9	decimal values
.	fractional delimiter; null token
=	barline; == double barline

Summary of ****synco** *Signifiers*

EXAMPLES

A sample document is given below:

```
**synco
*M2/4
*tb8
=1
0
0
0
0
=2
1.10
0
0
0
=3
0.69
0
*-
```

PERTINENT COMMANDS

The following Humdrum command produces ****synco** data as output:

synco	measure the degree of metric syncopation

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****synco**:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for ****synco**

SEE ALSO

barlines (2), ****metpos** (2), **synco** (4), ****timebase** (2)

REFERENCES

Longuet-Higgins, H. C., & Lee, C. S., "The perception of musical rhythms," *Perception*, Vol. 11 (1982) pp. 115-128.

REPRESENTATION

****takt** — beat-position representation

DESCRIPTION

Frequently, it is useful to represent temporal moments within a recurring cycle or pattern. In music, the best example of such recurring moments is the pattern of beats within a measure. In 3/4 meter, for example, a recurring cycle of beats occurs: 1-2-3-1-2-3 ... The ****takt** interpretation permits the representation of recurring moments within some temporal cycle. For most applications, the duration of the repeating cycle will be equivalent to the measure. However, other cycles are possible, such as phrase, sub-phrase, etc.

In ****takt**, integer values refer to beats, hence the value '2' refers to the second beat. Fractional values refer to moments within beats. For example, the value '3.5' refers to the second half of the third beat. In 4/4 meter, the value '4.75' refers to the last sixteenth of the measure. Up to two digits are permitted following the decimal point. Some decimal values are reserved to mean precise subdivisions. For example, the decimal values 'x.33' and 'x.67' are assumed to be precisely one-third and two-thirds respectively, i.e. equivalent to 'x.33333...' and 'x.66666...' Other fractional values are defined in the following table.

x.10	1/10
x.11	1/9
x.13	1/8
x.14	1/7
x.16	1/6
x.20	1/5
x.22	2/9
x.25	1/4
x.29	2/7
x.30	3/10
x.33	1/3, 3/9
x.38	3/8
x.40	2/5
x.43	3/7
x.44	4/9
x.50	1/2
x.56	5/9
x.57	4/7
x.60	3/5
x.63	5/8

x.67	2/3, 6/9
x.70	7/10
x.71	5/7
x.75	3/4
x.78	7/9
x.80	4/5
x.83	5/6
x.86	6/7
x.88	7/8
x.89	8/9
x.90	9/10

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**takt` data should be given names with the distinguishing '.tak' extension.

SIGNIFIERS

The following table summarizes the ****takt** mappings of signifiers and signifieds.

0-9	decimal values
.	fractional delimiter; null token
=	barlines
==	double barline

Summary of ****takt** *Signifiers*

EXAMPLES

A sample document is given below:

```
**kern   **metpos  **takt
*M4/4    *M4/4     *M4/4
*c:      *         *
=1       =1        =1
8r       1         1
16cc     4         1.5
16bn     5         1.75
8cc      3         2
8g       4         2.5
8a-      2         3
16cc     4         3.5
16b      5         3.75
8cc      3         4
8dd      4         4.5
=2       =2        =2
8g       1         1
16cc     4         1.5
16bn     5         1.75
8cc      3         2
8dd      4         2.5
16f      2         3
16g      5         3.25
[8a-     4         3.5
8a-]     3         4
16g      4         4.5
16f      5         4.75
=3       =3        =3
*-       *-        *-
```

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce ****takt** as output, or process ****takt** encoded data as input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****takt**:

meter signatures	`*M6/8`
tempo	`*MM96.3`

Tandem interpretations for ****takt**

SEE ALSO

barlines (2), ****date** (2), ****dur** (2), ****metpos** (2), ****ordo** (2), ****recip** (2), ****time** (2), **timebase** (4), ****Zeit** (2)

REPRESENTATION

**text — vocal text representation

DESCRIPTION

The **text representation permits the representation of sung or spoken language. Only languages transcribable into the Roman alphabet can be represented using **text. For non-Roman alphabets and non-language vocables, refer to the **IPA (2) representation.

The **text representation distinguishes three classes of data tokens: text-tokens, silence-tokens, and barlines.

Text-tokens are of three types: syllabic, melismatic, and polysyllabic. A *syllabic* text-token means that a single syllable is associated with a single moment (such as a single pitch). A *melismatic* text-token means that a single syllable is associated with several successive moments (such as more than one pitch). A *polysyllabic* text-token means that several syllables are associated with a single moment (such as a single pitch).

The simplest of these is the *melismatic* text-token — which consist merely of a single vertical bar (|). By itself, the vertical bar means that the previous syllable is sustained through the current moment. Syllabic and polysyllabic text-tokens are more complicated.

In the **text representation, text-tokens are syllable-oriented, so words must be broken-up into the component syllables. Words may be of three basic types: single-syllable words, multi-syllable words, and hyphenated words (e.g. the word "half-mast"). As a result, four types of syllables are distinguished: (1) a single-syllable word, (2) a word-initiating syllable, (3) a word-completing syllable, and (4) a mid-word syllable. In **text the hyphen (-) is used to signify syllable boundaries; the tilde (~) is used to signify boundaries between hyphenated words (necessarily also a syllable boundary). The following table illustrates the how these signifiers are used:

text	a single-syllable word
text-	a word-initiating syllable
-text	a word-completing syllable
-text-	a mid-word syllable
text~	a single-syllable word beginning a hyphenated multi-word
~text	a single-syllable word completing a hyphenated multi-word
~text~	a single-syllable word continuing a hyphenated multi-word
~text-	a word-initating syllable completing a hyphenated multi-word
-text~	a word-completing syllable — part of a hyphenated multi-word

A *syllabic* text-token consists of any one of the above tokens.

Occasionally, it is useful to encode several syllables or words for a single moment (or pitch)

— as in the case of *recitativo* passages. In the **text representation, such multi-word or polysyllabic text-tokens are encoded via Humdrum multiple-stops. That is, several words or syllables are encoded on the same data record, with each successive syllable separated by a space. The following encoding shows two syllables/words for each note:

```
**pitch   **text
C4        I saw
A4        a li-
G4        -ly beau-
F4        -ti- -ful ,
*-        *-
```

In the **text representation, punctuation marks — such as commas, periods, and exclamation marks — are segregated from the syllable texts by a space. The period character is represented by the asterisk (`*`) in order to avoid the illegal construction of a period (null token) in a multiple stop. Five remaining punctuation marks are represented literally in **text: comma (`,`), exclamation mark (`!`), question mark (`?`), colon (`:`), and the semicolon (`;`). When used in a melismatic context, an explicitly notated (printed) dash is represented by a double vertical bar (`||`). This signifies both the presence of a printed dash and the fact that the previous syllable continues to sound. In addition, double quotation marks (`"`) and parentheses (`()`) are also permitted. Once again, all such forms of punctuation must be segregated from the alphabetic text by the presence of a space. It is important to remember that periods are represented by asterisks.

The **text representation is intended to represent canonical language information rather than verbatim transcriptions of a printed score. Textual abbreviations and printing conventions such as *"etc.," "#"* and *"$"* must be expanded as: *et ce- -te- -ra, num- -ber,* and *dol- -lar* respectively (depending on the language). (If the user wishes to represent *text as printed,* an explicit **underlay or **overlay representation would be more appropriate.)

Where greater subtlety is required for pronunciation, refer to the **IPA (2) representation.

The original ASCII character set is regrettably hostile to non-English languages, especially with regard to accents. With the extended (international) ASCII set, French, German, and other accents are properly handled, but for English users, a given computer system may prove inflexible for non-English materials. In such cases, the **text representation provides the following aliases to designate common accents. Accent signifiers are assumed to modify the immediately preceding Roman letter.

/	acute accent
\	grave accent
^	circumflex
0	small superscript circle
1	macron
2	Umlaut
5	cidilla
6	hacek
7	enya (tilde)

Accent signifiers in ****text**

Musical phrasing can be explicitly represented in ****text** via the open brace ({) and closed brace (}) for phrase-beginnings and phrase-endings respectively. Slurs can be represented via the open bracket ([) and closed bracket (]). Stressed and unstressed syllables can be explicitly denoted by prepending the plus sign (+) or the underscore (_) respectively — at the beginning of the associated text.

A handful of vocables can be represented via ****text** including humming (signified by the upper-case letter 'M' by itself) and laughing (signified by the upper-case letter 'L' by itself). Tandem interpretation also permit encoding of the depth of vibrato and the degree of ennunciation. For example, shallow vibrato is indicated by `*vibrato1` whereas deep vibrato is indicated by `*vibrato9`. Sloppy ennunciation is indicated by `*ennun1` whereas highly precise ennunciation is indicated by `*ennun9`. Intermediate values can be used as needed.

Apart from text-tokens ****text** also permits the encoding of silence-tokens and barlines. Silence-tokens consist simply of the percent sign (%) appearing in isolation. This indicates the termination or absence of any vocal sound.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

Much vocal music is strophic in form — containing two or more "verses" sung to the same musical setting. Abbreviated representations may take advantage of the Humdrum *strophe* mechanism to avoid duplicate encoding of melodies, etc. See **strophe (4)**.

FILE TYPE

It is recommended that files containing predominantly `**text` data should be given names with the distinguishing '.txt' extension.

SIGNIFIERS

The complete system of signifiers used by ****text** is summarized in the following table.

A-Z	upper-case letters A to Z
a-z	lower-case letters a to z
%	silence token (character by itself)
L	laughing voice (character by itself)
M	humming voice (character by itself)
-	multi-syllable-word syllable boundary
~	hyphenated-word word separator; printed dash
|	melisma indicator (syllable sustain)
||	printed hyphen indicating melisma
.	null token (**not** period)
*	period (must be preceded by a space)
?	question mark (must be preceded by a space)
:	colon (must be preceded by a space)
;	semicolon (must be preceded by a space)
!	exclamation mark (must be preceded by a space)
(	open parenthesis
)	closed parenthesis
{	beginning of a phrase
}	end of a phrase
[	beginning of a slur
]	end of a slur
=	barline
==	double barline
/	acute accent
\	grave accent
^	circumflex
0	small circle
1	macron
2	Umlaut
5	cedilla
6	hacek
7	enya (tilde)

Summary of ****text** *Signifiers*

EXAMPLES

A sample document is given below:

```
**text
*English
=1
Once up-
||
-on a
.
time
|
there
=2
was a
lit- -tle
girl
.
whose
=3
*-
```

Notice that the first measure contains two Humdrum double-stops (`Once up-` and `-on a`). The double vertical bar (`||`) indicates that the word "upon" is explicitly divided by a notated hyphen. The word `time` is melismatic — spanning two time-events.

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce ****text** as output or process ****text** encoded data as input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****text**:

language indicator	`*Deutsch`
strophe indicator	`*strophe`
verse number	`*S/3`
choral text	`*ensemble`
solo text	`*solo`
meter signatures	`*M6/8`
underlay position	`*unter`
overlay position	`*ueber`
vibrato depth	`*vibrato7`
ennunciation	`*ennun5`

Tandem interpretations for ****text**

SEE ALSO

assemble (4), ****IPA** (2), **strophe** (4), **thru** (4)

REPRESENTATION

**time — relative elapsed time (in seconds)

DESCRIPTION

The **time representation is used to represent elapsed time (from some arbitrary moment deemed "time zero") to the onset of the current moment. A typical use for **time is to represent the elapsed time from the beginning of a work (in seconds). Data tokens in **time consist simply of numbers of seconds, with an optional decimal value. The **time representation has no provision for representing "hours" or "minutes".

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**time` data should be given names with the distinguishing '.tim' extension.

SIGNIFIERS

The following table summarizes the **time mappings of signifiers and signifieds.

0-9	decimal values
.	fractional second delimiter; null token
=	barlines
==	double barline

Summary of **time *Signifiers*

EXAMPLES

A sample document is given below:

```
**kern   **metpos   **takt   **time
*M4/4    *M4/4      *M4/4    *M4/4
*MM60    *MM60      *MM60    *MM60
*c:      *          *        *
=1       =1         =1       =1
8r       1          1        0
16cc     4          1.5      0.5
16bn     5          1.75     0.75
8cc      3          2        1
8g       4          2.5      1.5
8a-      2          3        2
16cc     4          3.5      2.5
16b      5          3.75     2.75
8cc      3          4        3
8dd      4          4.5      3.5
=2       =2         =2       =2
8g       1          1        4
16cc     4          1.5      4.5
16bn     5          1.75     4.75
8cc      3          2        5
8dd      4          2.5      5.5
16f      2          3        6
16g      5          3.25     6.25
4a-      4          3.5      6.5
*-       *-         *-       *-
```

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce **time as output, or process **time encoded data as input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with **time:

MIDI channel	`*Ch1`
meter signatures	`*M6/8`
tempo	`*MM96.3`

Tandem interpretations for **time

SEE ALSO

barlines (2), ****date** (2), ****dur** (2), ****metpos** (2), ****ordo** (2), ****recip** (2), ****takt** (2), ****Zeit** (2)

REPRESENTATION

****Tonh** — German Tonhöhe (pitch) notation

DESCRIPTION

The ****Tonh** representation permits the encoding of Western musical pitches using the common German system for pitch naming. This system is similar to the English system of pitch designations with the exceptions noted below.

The ****Tonh** representation distinguishes three types of data tokens: pitches, rests, and barlines. Pitches are encoded using pitch letter-name, accidental, and octave designation. In addition, ****Tonh** provides limited capabilities for representing phrasing and slurs.

Pitch tokens consist of three logical parts — without any intervening spaces. The first part is the pitch letter-name; only the upper-case letters A, B, C, D, E, F, G, H, and S are permitted. The second part is an optional accidental: the suffix 'is' designates a sharp, while the suffix 'es' designates a flat — hence 'Cis' for C-sharp and 'Ges' for G-flat. Suffixes may be repeated for double and triple sharps and flats. Special exceptions include the following: 'B' for B-flat, 'H' for B-natural, 'Heses' for B double-flat (rather than 'Bes'), and 'As' and 'Es' rather than 'Aes' or 'Ees'. In addition, 'S' may be used as an alias for 'Es' (E-flat).

The third part of a pitch token is the octave designation. The number 4 is used to designate all pitches between middle C and the H a major seventh above, inclusive. Octave numbers are incremented by one for each successively higher octave, and are decremented by one for each successively lower octave. Negative octave numbers are not permitted, so the lowest pitch in the ****Tonh** representation is C0 (16.35 Hz). Only a single octave digit is permitted, so the highest ****Tonh** pitch is H9 (15,804 Hz).

Once again, no intervening spaces are permitted within a single note. Notice that the order of signifiers is important for pitch encodings. Pitch letter-name is followed by one or more accidentals (if appropriate), followed by an octave designation.

Several notes may be encoded concurrently in a single spine by using the Humdrum multiple-stop convention: notes within multiple-stops are separated by single spaces. The following example encodes a C-minor chord as four pitches in two ****Tonh** spines — each spine containing a double-stop.

```
**Tonh   **Tonh
C4 Es4   G4 C5
*-       *-
```

Pitch tokens may be modified by the presence of additional signifiers. The open brace '{' denotes the beginning of a phrase. The closed brace '}' denotes the end of a phrase. The open parenthesis '(' denotes the beginning of a slur. The closed parenthesis ')' denotes the end of a slur. The semicolon ';' denotes a pause.

Rests tokens are denoted by the lower-case letter 'r'.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**Tonh` data should be given names with the distinguishing '.tnh' extension.

SIGNIFIERS

The following table summarizes the ****Tonh** mappings of signifiers and signifieds.

0-9	octave designation, where C4 equals middle C;
A-H	absolute pitches letter name
es	flat
eses	double flat ('eseses' for triple flat)
is	sharp
isis	double sharp
Heses	B double-flat (rather than 'Bes')
As	A-flat (rather than 'Aes')
Es	E-flat (rather than 'Ees')
S	alias for 'Es'
n	natural
r	rest
=	barline; == double barline
(	slur start
)	slur end
{	phrase mark (start)
}	phrase mark (end)
;	pause sign

Summary of ****Tonh** *Signifiers*

EXAMPLES

A sample document is given below:

```
!! Anton Webern
!! Klavierstück, opus posthumous
!! Im Tempo eines Menuetts
**Tonh
=2
(Cis4
Dn5)
Es3
r
Fis3
F4
E2
=3
Cn6
B2 A3
H4
Gis2 G3
Cis4
D2 S3
=4
*-
```

PERTINENT COMMANDS

The following Humdrum commands accept ****Tonh** encoded data as inputs:

cents	translate ****Tonh** to ****cents**
deg	translate ****Tonh** to ****deg**
degree	translate ****Tonh** to ****degree**
freq	translate ****Tonh** to ****freq**
hint	calculate harmonic intervals from ****Tonh** input
kern	translate ****Tonh** to ****kern**
mint	calculate melodic intervals from ****Tonh** input
pc	translate ****Tonh** pitch to ****pc**
pitch	translate ****Tonh** pitch to ****pitch**
semits	translate ****Tonh** pitch to numerical ****semits**
solfa	translate ****Tonh** pitch to ****solfa**
solfg	translate ****Tonh** pitch to ****solfg**
trans	transpose ****Tonh** score
vox	determine active and inactive voices in a Humdrum file

The following Humdrum command produces ****Tonh** data as output:

tonh	translates ****cents, **degree, **fret, **freq, **kern,**

****MIDI**, ****pitch**, ****semits**, ****solfg**, and ****specC** to ****Tonh**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****Tonh**:

meter signatures	`*M6/8`
key signatures	`*k[f#c#]`
key	`*c#:`
tempo	`*MM96.3`

Tandem interpretations for ****Tonh**

SEE ALSO

barlines (2), ****cents** (2), ****deg** (2), ****degree** (2), ****freq** (2), ****hint** (2), **hint** (4), ****kern** (2), ****mint** (2), **mint** (4), ****pc** (2), **pc** (4), ****pitch** (2), **pitch** (4), ****semits** (2), ****solfa** (2), ****solfg** (2), **solfg** (4), ****specC** (2), **tonh** (4)

REPRESENTATION

****URrhythm** — represent Johnson-Laird beat prototypes for a passage

DESCRIPTION

The ****URrhythm** representation encodes beat "prototypes" (Ur-rhythms) evident in a musical passage. The representation is based on Johnson-Laird's theory of rhythmic prototypes (see REFERENCES below).

The ****URrhythm** representation characterizes each beat in a passage as belonging to one of three "beat types": *Note* (N), *Syncopation* (S), or *Other* (O). Only principle beats are characterized in this way. Hence, in 3/4 or 9/8 meters, three beats are characterized for each complete measure. Similarly, in 4/2 and 12/16, four beats are characterized for each complete measure. Johnson-Laird's rhythmic-prototype theory can be applied only to musical passages conforming to regular meters (simple and compound, duple, triple and quadruple).

A "Note" (signified in the output by the letter 'N') is defined as a beat that coincides with a note onset.

A metric "Syncopation" (signified by the letter 'S') is defined as arising when no note-onset happens on a beat whose metric position is more important 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.

Metrically syncopated beats can happen only after the first note onset; subsequent syncopated moments 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).

Barlines are represented using the "common system" for barlines — see **barlines (2).**

FILE TYPE

It is recommended that files containing predominantly `**URrhythm` data should be given names with the distinguishing '.urr' extension.

SIGNIFIERS

The following table summarizes the ****URrhythm** mappings of signifiers and signifieds.

N	"note" beat prototype
O	"other" beat prototype
S	"syncopation" beat prototype
.	null token
=	barline; == double barline
0-9	measure numbers

Summary of ****URrhythm** *Signifiers*

EXAMPLES

A sample document is given below:

```
**URrhythm
*M4/4
*tb8
N
.
N
.
S
.
=1
N
.
O
.
S
.
O
.
=2
S
.
S
.
O
.
N
.
*-
```

PERTINENT COMMANDS

The following Humdrum command produces ****URrhythm** data as output:

urrhythm characterize the rhythmic prototypes in a passage

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****URrhythm**:

meter signatures	`*M6/8`
tempo	`*MM96.3`
timebase	`*tb32`

Tandem interpretations for ****URrhythm**

SEE ALSO

barlines (2), ****metpos** (2), **metpos** (4), **urrhythm** (4), ****synco** (2), **synco** (4), ****timebase** (2)

REFERENCES

Philip Johnson-Laird, "Rhythm and meter: A theory at the computational level," *Psychomusicology*, Vol. 10 (1991) pp. 88-106.

Jasba Simpson & David Huron, "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.

REPRESENTATION

**vox# — representation of number of concurrently active voices

DESCRIPTION

The **vox# representation permits the representation of the number of notes sounding together at successive moments in time. The **vox# representation distinguishes just two types of data tokens: number-tokens, and barlines. Each **vox# number-token consists of a single integer value indicating the total number of simultaneously sounding voices.

Barlines are represented using the "common system" for barlines — see **barlines (2)**.

FILE TYPE

It is recommended that files containing predominantly `**vox#` data should be given names with the distinguishing '.vox' extension.

SIGNIFIERS

The following table summarizes the **vox# mappings of signifiers and signifieds.

0-9	integer values; measure numbers
=	barline; == double barline
;	pause (for barlines)

Summary of **vox#** *Signifiers*

EXAMPLES

A sample document is given below:

```
**kern   **kern   **kern   **kern         **kern         **vox#
=18      =18      =18      =18            =18            =18
8r       8B       4.g      8g             8dd)           4
8BBn     8B       .        8dd            8gg            5
8BB      8B       .        8dd            8gg            5
8BB      8B       8d       8g             8ff            5
=19      =19      =19      =19            =19            =19
4C       4c       4e-      4c 4g 4ee-     4ee-           7
4r       4r       4r       4r             4r             0
=20      =20      =20      =20            =20            =20
4AA-     4AA-     4a-      4a- 4f# 4cc    4a- 4f# 4cc    9
8.r      8.r      8.r      8.r            8.r            0
=21      =21      =21      =21            =21            =21
*        *        *        *              *^             *
4GG      4GG      4G       4g 4d 4bn      4G 4d 4bn      2gg;      10
4r       4r       4r       4r             4r             .         1
*-       *-       *-       *-             *-             *-        *-
```

PERTINENT COMMANDS

The following Humdrum command produces ****vox#** data as output:

vox determine number of concurrently active voices for ****cbr**, ****cents**, ****cocho**, ****deg**, ****degree**, ****freq**, ****kern**, ****pc**, ****pitch**, ****semits**, ****solfa**, ****solfg**, ****specC**, and ****Tonh**

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****vox#**:

meter signatures	`*M6/8`

Tandem interpretations for ****vox#**

SEE ALSO

barlines (2), ****barks** (2), ****cbr** (2), ****cents** (2), ****cocho** (2), ****deg** (2), ****degree** (2), ****freq** (2), ****kern** (2), ****pc** (2), ****pitch** (2), ****semits** (2), ****solfa** (2), ****solfg** (2), ****specC** (2), ****Tonh** (2), **vox** (4)

REPRESENTATION

****Zeit** — absolute period of time

DESCRIPTION

The ****Zeit** representation is used to represent spans of time, such as the life-span of a composer, or the chronology of a rehearsal. The syntax for ****Zeit** is nearly identical to the ****date** representation. Since ****Zeit** represents a span of time, two date tokens must be specified and separated by a dash (-). For example, the ****Zeit** data token `1770/-1827/` represents the period from 1770 to 1827.

The ****Zeit** representation includes all of the features of ****date** for signifying approximation (~), uncertainty (?), approximate value (x), uncertain value (z), as well as between-range (^), either-or (|), prior (<) and after (>) boundaries. Either one or both of the dates specified in a ****Zeit** data token may encode complex degrees of approximation or uncertainty.

Conceptually, ****Zeit** data tokens consist of two ****date** "sub-tokens." Zeit tokens are encoded according to the following basic syntax:

year/month/day/hour:minute:second.decimal-year/month/day/hour:minute:second.decimal

The data tokens making-up the ****Zeit** information may be encoded in full, or may consist of isolated elements or parts. The following table shows the most succinct ways of encoding single date values within sub-tokens in ****Zeit**:

`.11`	eleven one-hundredths of a second
`11`	11th second
`11:`	11th minute
`11::`	11 o'clock
`11/`	A.D. 11
`/11`	November
`//11`	11th day of the month

Examples of **date** *sub-components*

Notice that if a single numerical value appears, it is interpreted as *seconds*; if a single value appears followed by a slash, it is interpreted as a *year*; if a single value appears followed by a colon, it is interpreted as a *minutes*. Days and hours require two leading or two trailing delimiters respectively. In general, abbreviated forms of date sub-tokens tend to favor the two extremes of time: seconds and years. These are the time frames that are typically of greatest interest to music scholars.

The ****Zeit** representation makes use of the Gregorian calendar and the 24-hour clock. Dates prior to the year 1 A.D. can be specified by prepending a minus sign to the year.

The ****Zeit** representation provides three distinct means for representing approximate moments. It also provides two independent means for representing uncertainty, as well as mechanisms for representing time boundaries (prior to ...; after ...). For the appropriate representation syntax refer to ****date (2).**

FILE TYPE

It is recommended that files containing predominantly `**Zeit` data should be given names with the distinguishing '.zt' extension.

SIGNIFIERS

The following table summarizes the ****Zeit** mappings of signifiers and signifieds.

-	sub-token separator (from-to); (must appear in each ****Zeit** data token)
0-9	decimal values
@	year B.C. rather than A.D.
/	year-month, month-day and day-hour delimiter
:	hour-minute and minute-second delimiter
.	fractional second delimiter; null token
?	date uncertain
z	value uncertain
~	date approximate
x	value approximate
<	sometime prior to
>	sometime after
^	"between" conjunction
|	"or" conjunction

Summary of ****Zeit** *Signifiers*

EXAMPLES

Several examples of ****Zeit** data tokens are identified below:

**Zeit tokens	meaning
`1939/-1945/`	From 1939 to 1945.
`1817/06/02/-1817/6/15`	From June 2nd to 15th, 1817.
`1817/6/02/-1817/06/15`	From June 2nd to 15th, 1817.
`///10::-///11::`	From 10 AM to 11 AM.
`10::-11::`	From 10 AM to 11 AM.
`~10::-~11::`	From approximately 10 AM to approximately 11 AM.
`>22::-<23::`	From sometime after 10 PM to sometime before 11 PM.
`:11:51-:12:35`	From 11 minutes 51 seconds to 12 minutes 35 seconds.
`.001-.008`	From 1 millisecond to 8 milliseconds.
`23.8-41.3`	From 23.8 seconds to 41.3 seconds.
`//12/31-//1/1`	From December 31st to New Years' Day.
`<?1231///-1283/3/9`	From before perhaps 1231 to March 9th 1283.
`<1724/2//-1724/4z/2`	From before Feb. 1724 to April (?) 2nd 1724.
`1848/|1849/-1851/`	From 1848 or 1849 to 1851.
`/5/9/^/5/11/-//8/23`	Starting sometime between May 9th and 11th ending August 23rd.

Examples of ****Zeit** *Tokens*

The following examples illustrate the use of the `**Zeit` representation:

```
**Zeit               **maker
?1644/-1737/12/18    Stradivari, Antonio
1794/4/9-1881/11/25  Boehm, Theobald
1797/2/15-1871/2/7   Steinweg, Heinrich
1814/11/6-1894/2/4   Sax, Adolphe
*-                   *-
```

```
**Zeit                 **recording log
/4/9:20:18-/4/9:20:20  Aria - Take #1
/4/9:20:20-/4/9:20:22  Aria - Take #2
/4/9:20:23-/4/9:20:25  Aria - Take #3
/4/9:20:25-/4/9:20:27  Var.1 - Take #1
*-                     *-
```

```
**Zeit       **section
0:0-0:23     Introduction
0:23-1:58    Exposition
1:58.3-3:22  Development
3:22-4:51    Recapitulation
4:52-5:04    Coda
*-           *-
```

```
**Zeit          **style
~1450/-~1600/   Renaissance
~1600/-~1750/   Baroque
~1775/-~1825/   Classicism
~1800/-~1900/   Romanticism
*-              *-
```

PERTINENT COMMANDS

Currently, no special-purpose Humdrum commands produce ****Zeit** as output, or process ****Zeit** encoded data as input.

TANDEM INTERPRETATIONS

The following tandem interpretations can be used in conjunction with ****Zeit**:

meter signatures	`*M6/8`

Tandem interpretations for ****Zeit**

SEE ALSO

****date** (2), ****dur** (2), ****metpos** (2), ****ordo** (2), ****recip** (2), ****takt** (2), ****time** (2)

Section 3

Tandem Interpretation Reference

Documentation Style

This section of the *Reference Manual* describes various pre-defined tandem interpretations that are used in conjunction with some of the representations described in Section 2. Tandem interpretations provide additional contextual information for a given representation. Tandem interpretations are denoted by a single leading asterisk character (beginning in the first column of a line and spine) followed immediately by an interpretation *keyword.* (By contrast, *exclusive interpretations* are denoted by two leading asterisks.) Although only one exclusive interpretation can be active at a given moment for a given spine, several tandem interpretations may be active concurrently in a given spine. Tandem interpretations may appear at any point in a file, but they must be preceded by an exclusive interpretation specifying the type of representation encoded.

Each entry in this section of the documentation includes a description of the scope of the tandem representation, a syntax for the interpretation, and examples of use. Descriptions of the corresponding representations may be found in Section 2 of this *Reference Manual.*

Each reference entry contains information identifying the name and purpose of the tandem interpretation and a summary description of mappings between signifiers and signifieds. The standard order of documentation sections is as follows: (1) representation, (2) description, (3) signifiers, (4) examples, (5) see also, (6) warnings, (7) note, (8) reference, and (9) proposed modifications

REPRESENTATION

all intervals — all harmonic intervals designator

DESCRIPTION

The **all intervals** tandem interpretation is used in conjunction with the `**hint` harmonic interval representation to indicate that the representation includes all possible permuted intervals. For example, a sonority consisting of the pitches C4, E4, G4, and C5 would produce an exhaustive interval content including M3, m3, P4, P4, m6 and P8.

The `*all` tandem interpretations consist simply of a single asterisk, followed by the keyword `all`.

EXAMPLES

An example of the use of `*all` is given below:

```
**pitch       **hint       **hint
*             *            *all
C4            -            -
C4 E4         M3           M3
C4 E4 G4      M3 m3        M3 m3 P5
C4 E4 G4 C5   M3 m3 P4     M3 m3 P4 P5 m6 P8
*-            *-           *-
```

SEE ALSO

****hint** (2), **hint** (4)

REPRESENTATION

clefs — clef designation

DESCRIPTION

The **clef** tandem interpretation permits the encoding of notated clefs for a Humdrum representation.

Three types of clefs can be represented: G-clefs, F-clefs, and C-clefs. Each clef may be placed on any line in a multi-line staff. The common treble staff locates the G-clef on the second line from the bottom, while the common bass staff locates the F-clef on the fourth line from the bottom. In addition, octave and double-octave transpositions can be represented.

Clef tandem interpretations consist of a single asterisk, followed by the keyword `clef`, followed by an upper-case letter indicating the type of clef, followed by one or more octave transposition signifiers (`^` or `v`), followed by a number indicating the designated line. followed by a number indicating the staff-line designated by the clef. Line numbers are counted beginning at the bottom of the staff. The absence of any clef indication may be explicitly represented by the '`X`' clef designator — as in `clefX`. Notice that clef tandem interpretations do not assume the number of lines in the staff. Hence a C-clef appearing on the third line (from the bottom) of a four-line staff would be encoded as `*clefC3`.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for clefs.

`clef`	clef keyword
`C`	C-clef signifier
`F`	F-clef signifier
`G`	G-clef signifier
`X`	no clef signifier
`1-9`	line number designators
`^`	8va treble
`v`	8va bassa
`^^`	double octave treble
`vv`	double octave bass

Summary of **clef** *Signifiers*

EXAMPLES

Several examples of clef indications are given below:

`*clefG2`	treble clef
`*clefF4`	bass clef
`*clefC3`	alto clef
`*clefC4`	tenor clef
`*clefG1`	soprano clef
`*clefX`	no clef
`*clefGv2`	treble clef, 8va bassa

Examples of **clef** *Interpretations*

SEE ALSO

staff (3), **staff lining** (3)

REPRESENTATION

ensemble — designation of the number of instruments/voices

DESCRIPTION

The **ensemble** tandem interpretation permits the encoding of the number of musical instruments of voices performing a given Humdrum spine.

Five types of ensemble interpretations are distinguished. The `*solo` interpretation indicates that a given spine is to be performed by a single instrument or voice, and that the part is of a *foreground* character. The `*ensemb` interpretation indicates that a given spine is to be performed by several instruments or voices. The `*ripien` interpretation indicates that a given spine is performed by an instrument of the *ripieno.* The `*conct` interpretation indicates that a given spine is performed by an instrument of the *concertino.* The fifth type of ensemble-related interpretation allows the precise specification of the number of instruments or voices. This tandem interpretation consists of a single asterisk, followed by the lower-case letter 'a', followed by a number. For example, `*a2` represents an ensemble indication of two instruments or voices. Similarly, the tandem interpretation `*a6` indicates six concurrent instruments performing the specific spine. Approximation can be indicated by appending the tilde character, such as `*a12~` — meaning approximately 12 instruments.

EXAMPLES

Several examples of ensemble indications are given below:

`*solo`	single instrument or voice
`*ensemb`	multiple instruments or voices
`*a1`	single instrument or voice
`*a2`	two instruments or voices
`*a14`	fourteen instruments or voices
`*conct`	instrument of the concertino
`*ripien`	instrument of the ripieno

Examples of **ensemble** *Interpretations*

SEE ALSO

instrument (3)

REPRESENTATION

expansion lists — expansion list designations

DESCRIPTION

An **expansion list** is a tandem interpretation that indicates how an *abbreviated format* Humdrum file may be rearranged or expanded to a full-length or *through-composed format.*

Expansion lists are found only in abbreviated format files. An expansion list contains an ordered list of Humdrum section labels identifying the order (including possible repetitions) of sections when the file is passed to the **thru** command. Expansion lists are useful for encoding Da Capo, Dal Segnos, and other repetition notational devices. Expansion list are also useful for encoding alternative versions of the organization of a work.

Expansion lists consist of a single asterisk, followed by the greater-than sign (>), followed by an optional keyword, followed by an open square bracket, followed by a list of section labels (each separated by a comma), followed by a closed square bracket. Consider the following expansion list:

```
*>[verse1,refrain,verse2,refrain]
```

This list indicates that the file in which it is embedded is an abbreviated format Humdrum file that contains (at least) three sections, labelled `verse1`, `verse2`, and `refrain`. When the file is expanded using the **thru** command, the `refrain` section will be repeated following each verse.

The following example illustrates two expansion lists, each of which is labelled. Expansion-list labels are called *versions.* In this example, the first and second versions are `Gould82` and `Landowska` respectively.

```
*>Gould82[A,A,B]
*>Landowska[A,A,B,B]
```

These expansion lists might encode different interpretations of the repeats in a rounded binary form. (When the **thru** command is invoked, the user can specify which *version* is required, and the appropriate through-composed expansion will be output.)

SIGNIFIERS

The *version* keywords for expansion lists may contain any sequence of zero or more printable ASCII characters with the exception of the tab character and the open square bracket. Immediately following the version keyword is an open square bracket. An expansion list must end with a square bracket. Within the square brackets zero or more section labels may be encoded, separated by commas.

EXAMPLES

Several examples of expansion list tandem interpretations are given below:

```
*>sonata allegro[intro,exposition,development,recapitulation]
*>[minuet,trio,minuet]
*>Rondo[A,B,A,C,A,D,A,B,A]
*>rehearsal order[mm. 218-252,mm. 184-191,mm. 1-48]
*>concert[Stamitz,Martinu,Alkan,De Falla]
*>subject18[stimulus7,stimulus9,stimulus4,stimulus2]
```

Examples of **expansion list** *Interpretations*

SEE ALSO

section labels (3), **thru** (3), **thru** (4), **yank** (4)

REPRESENTATION

fret tuning — fretted isntrument tuning information

DESCRIPTION

Three tandem interpretations permit the detailed encoding of tuning information for fretted instruments.

The absolute tuning is specified using the `*AT:` tandem interpretation. The relative tuning of the open strings is specified using the `*RT:` tandem interpretation. The tuning of the fret positions is specified using the `*FT:` tandem interpretation.

The `*AT:` interpretation uses `**pitch`-type pitch designations (including cents deviation) to encode the absolute pitch of the lowest string. (See EXAMPLES.)

The `*RT:` interpretation encodes the relative tuning of each open string by specifying the number of semitones above the lowest string. Successive courses are delineated by colons, and strings within courses are delineated by a comma. In addition to unbounded *scordatura* tuning, non-integer semitones may be encoded, thus permitting unorthodox temperaments. (See EXAMPLES.)

The `*FT:` interpretation encodes the relative tuning of successive frets along the fret-board, in semitones. Once again, non-integer semitones are permitted.

For a more detailed description of fretted instrument tuning interpretations, refer to the entry for `**fret` (Section 2).

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for fret tuning.

AT	absolute tuning keyword
RT	relative tuning keyword
FT	fret-board tuning keyword
:	course delimiter
,	string delimiter
A-G	pitch of lowest string (for *AT: only)
#	sharp accidental, for pitch of lowest string (for *AT: only)
b	flat accidental, for pitch of lowest string (for *AT: only)
0-9	semitone numbers; octave number; cents deviation
.	decimal point
-	cents deviation (for *AT: only)
+	cents deviation (for *AT: only)

Summary of **fret tuning** *Signifiers*

EXAMPLES

A number of examples of fret tuning indications are given in Section 2; refer to the entry for the `**fret` representation.

SEE ALSO

****fret** (2), ****pitch** (2)

REPRESENTATION

harmonic number — harmonic number designation

DESCRIPTION

The **harmonic number** tandem interpretation allows the encoding of given harmonic number for a Humdrum representation.

Harmonic number tandem interpretations consist of a single asterisk, followed by the keyword letter `H`, followed by an upper-case letter indicating the type of clef, followed by a whole number indicating the harmonic number. Harmonic 'zero' is illegal.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for harmonic number.

H	harmonic number keyword
0-9	number designators

Summary of **harmonic number** *Signifiers*

EXAMPLES

Several examples of harmonic number indications are given below:

`*H2`	second harmonic
`*H20`	twentieth harmonic
`*H02`	second harmonic

Examples of **harmonic number** *Interpretations*

SEE ALSO

****freq** (2), ****spect** (2)

REPRESENTATION

instrument — instrument or voice designation

DESCRIPTION

Instrument tandem interpretations are used to identify the instrumentation pertaining to a specified spine. The word "instrument" is used in a broad sense and embraces vocal qualities and types as well as mechanical sound makers.

Instrument tandem interpretations consist of a single asterisk, followed by the single upper-case letter `I`, followed by a lower-case instrument keyword.

Separate spines may be encoded for each instrument in a score, but it is common for two or more instruments to perform precisely the same line — such as the contrabass and violoncello parts. Instrument tandem interpretations are normally encoded "cumulatively;" that is, if more than one tandem interpretation appears in a spine, then this instrument is added to any existing instruments performing the spine.

SIGNIFIERS

The following set of tables list currently defined instrument tandem interpretations according to instrument category or type. Where appropriate, instrument names are also given for selected non-English languages.

Voice Range

`*Isoprn`	soprano
`*Imezzo`	mezzo soprano
`*Icalto`	contralto
`*Itenor`	tenor
`*Ibarit`	baritone
`*Ibass`	bass

Voice Quality

`*Ivox`	generic (undesignated) voice
`*Ifeme`	female voice
`*Imale`	male voice
`*Infant`	child's voice

`*Irecit`	recitativo
`*Ilyrsp`	lyric soprano
`*Idrmsp`	dramatic soprano
`*Icolsp`	coloratura soprano
`*Ialto`	alto
`*Ictenor`	counter-tenor
`*Iheltn`	Heldentenor, tenore robusto
`*Ilyrtn`	lyric tenor
`*Ibspro`	basso profondo
`*Ibscan`	basso cantante
`*Ifalse`	falsetto
`*Icastr`	castrato

String Instruments

`*Iarchl`	archlute; *archiluth* (Fr.); *liuto attiorbato/arcileuto/arciliuto* (It.)
`*Iarpa`	harp; *arpa* (It.), *arpa* (Span.)
`*Ibanjo`	banjo
`*Ibiwa`	biwa
`*Ibguit`	electric bass guitar
`*Icbass`	contrabass
`*Icello`	violoncello
`*Icemba`	harpsichord; *clavecin* (Fr.); *Cembalo* (Ger.); *cembalo* (It.)
`*Icetra`	cittern; *cistre/sistre* (Fr.); *Cither/Zitter* (Ger.); *cetra/cetera* (It.)
`*Iclavi`	clavichord; *clavicordium* (Lat.); *clavicorde* (Fr.)
`*Idulc`	dulcimer
`*Ieguit`	electric guitar
`*Iforte`	fortepiano
`*Iguitr`	guitar; *guitarra* (Span.); *guitare* (Fr.); *Gitarre* (Ger.); *chitarra* (It.)
`*Ihurdy`	hurdy-gurdy; variously named in other languages
`*Iliuto`	lute; *lauto, liuto leuto* (It.); *luth* (Fr.); *Laute* (Ger.)
`*Ikit`	kit; variously named in other languages
`*Ikokyu`	kokyū (Japanese spike fiddle)
`*Ikomun`	kŏmun'go (Korean long zither)
`*Ikoto`	koto (Japanese long zither)
`*Imando`	mandolin; *mandolino* (It.); *mandoline* (Fr.); *Mandoline* (Ger.)
`*Ipiano`	pianoforte
`*Ipipa`	Chinese lute
`*Ipsalt`	psaltery (box zither)
`*Iqin`	qin, ch'in (Chinese zither)
`*Iquitr`	gittern (short-necked lute); *quitarre* (Fr.); *Quinterne* (Ger.)
`*Irebec`	rebec; *rebeca* (Lat.); *rebec* (Fr.); *Rebec* (Ger.)
`*Isarod`	sarod

`*Ishami`	shamisen (Japanese fretless lute)
`*Isitar`	sitar
`*Itambu`	tambūrā
`*Itanbr`	tanbur
`*Itiorb`	theorbo; *tiorba* (It.); *tèorbe* (Fr.); *Theorb* (Ger.)
`*Iud`	ūd
`*Iukule`	ukulele
`*Ivina`	vīnā
`*Iviola`	viola; *alto* (Fr.); *Bratsche* (Ger.)
`*Iviolb`	bass viola da gamba; *viole* (Fr.); *Gambe* (Ger.)
`*Iviold`	viola d'amore; *viole d'amour* (Fr.); *Liebesgeige* (Ger.)
`*Iviol n`	violin; *violon* (Fr.); *Violine* (Ger.); *violino* (It.)
`*Iviols`	treble viola da gamba; *viole* (Fr.); *Gambe* (Ger.)
`*Ivio lt`	tenor viola da gamba; *viole (Fr.); Gambe (Ger.)*
`*Izithr`	zither; *Zither* (Ger.); *cithare* (Fr.); *cetra da tavola* (It.)

Wind Instruments

`*Iaccor`	accordion; *accordéon* (Fr.); *Akkordeon* (Ger.)
`*Iarmon`	harmonica; *armonica* (It.)
`*IbagpS`	bagpipe (Scottish)
`*IbagpI`	bagpipe (Irish)
`*Icalam`	chalumeau; *calamus* (Lat.); *kalamos* (Gk.)
`*Icalpe`	calliope
`*Icangl`	english horn; *cor anglais* (Fr.)
`*Ichlms`	soprano shawm, chalmeye, shalme, etc.; *chalemie* (Fr.); *ciaramella* (It.)
`*Ichlma`	alto shawm, chalmeye, shalme, etc.
`*Ichlmt`	tenor shawm, chalmeye, shalme, etc.
`*Iclars`	soprano clarinet (in either B-flat or A); *clarinetto* (It.)
`*Iclarp`	piccolo clarinet
`*Iclara`	alto clarinet
`*Iclarb`	bass clarinet (in B-flat)
`*Icor`	horn; *cor* (Fr.); *corno* (It.); *Horn* (Ger.)
`*Icornm`	cornemuse; French bagpipe
`*Icorno`	cornett (woodwind instr.); *cornetto* (It.); *cornaboux* (Fr.); *Zink* (Ger.)
`*Icornt`	cornet (brass instr.); *cornetta* (It.); *cornet à pistons* (Fr.); *Cornett* (Ger.)
`*Ictina`	concertina; *concertina* (Fr.); *Konzertina* (Ger.)
`*Ifagot`	bassoon; *fagotto* (It.)
`*Ifag_c`	contrabassoon; *contrafagotto* (It.)
`*Ifife`	fife
`*Iflt`	flute; *flauto* (It.); *Flöte* (Ger.); *flûte* (Fr.)
`*Iflt_a`	alto flute
`*Iflt_b`	bass flute
`*Ifltds`	soprano recorder; *flûte à bec, flûte douce* (Fr.); *Blockflöte* (Ger.); *flauto dolce* (It.)

`*Ifltdn`	sopranino recorder
`*Ifltda`	alto recorder
`*Ifltdt`	tenor recorder
`*Ifltdb`	bass recorder
`*Iflugh`	flugelhorn
`*Ihichi`	hichiriki (Japanese double reed used in gagaku)
`*Ikrums`	soprano crumhorn; *Krummhorn/Krumbhorn* (Ger.); *tournebout* (Fr.)
`*Ikruma`	alto crumhorn
`*Ikrumt`	tenor crumhorn
`*Ikrumb`	bass crumhorn
`*Inokan`	nōkan (Japanese flute for the nō theatre)
`*Ioboe`	oboe; *hautbois* (Fr.); *Hoboe, Oboe* (Ger.): *oboe* (It.)
`*IoboeD`	oboe d'amore
`*Iocari`	ocarina
`*Iorgan`	pipe organ; *organum* (Lat.); *organo* (It.); *orgue* (Fr.); *Orgel* (Ger.)
`*Ipanpi`	panpipe
`*Ipicco`	piccolo
`*Iporta`	portative organ
`*Irackt`	racket; *Rackett* (Ger.); *cervelas* (Fr.)
`*Ireedo`	reed organ
`*Isarus`	sarrusophone
`*IsaxN`	sopranino saxophone (in E-flat)
`*IsaxS`	soprano saxophone (in B-flat)
`*IsaxA`	alto saxophone (in E-flat)
`*IsaxT`	tenor saxophone (in B-flat)
`*IsaxR`	baritone saxophone (in E-flat)
`*IsaxB`	bass saxophone (in B-flat)
`*IsaxC`	contrabass saxophone (in E-flat)
`*Ishaku`	shakuhachi
`*Isheng`	mouth organ (Chinese)
`*Isho`	mouth organ (Japanese)
`*IsxhS`	soprano saxhorn (in B-flat)
`*IsxhA`	alto saxhorn (in E-flat)
`*IsxhT`	tenor saxhorn (in B-flat)
`*IsxhR`	baritone saxhorn (in E-flat)
`*IsxhB`	bass saxhorn (in B-flat)
`*IsxhC`	contrabass saxhorn (in E-flat)
`*Itromt`	tenor trombone; *trombone* (It.); *trombone* (Fr.); *Posaune* (Ger.)
`*Itromb`	bass trombone
`*Itromp`	trumpet; *tromba* (It.); *trompette* (Fr.); *Trompete* (Ger.)
`*Ituba`	tuba
`*Izurna`	zūrnā

1 s

1 l.

Percussion Instruments

`*Ibdrum`	bass drum (kit)
`*Icampn`	bell; *campana* (It.); *cloche* (Fr.); *campana* (Span.)
`*Icaril`	carillon
`*Icasts`	castanets; *castañetas* (Span.); *castagnette* (It.)
`*Ichime`	chimes
`*Iclest`	celesta; *céleste* (Fr.)
`*Icrshc`	crash cymbal (kit)
`*Ifingc`	finger cymbal
`*Iglock`	glockenspiel
`*Igong`	gong
`*Imarac`	maracas
`*Imarim`	marimba
`*Ipiatt`	cymbals; *piatti* (It.); *cymbales* (Fr.); *Becken* (Ger.); *kymbos* (Gk.)
`*Iridec`	ride cymbal (kit)
`*Isdrum`	snare drum (kit)
`*Ispshc`	splash cymbal (kit)
`*Isteel`	steel-drum, tinpanny
`*Itabla`	tablā
`*Itambn`	tambourine, timbrel; *tamburino* (It.); *Tamburin* (Ger.)
`*Itimpa`	timpani; *timpani* (It.); *timbales* (Fr.); *Pauken* (Ger.)
`*Itom`	tom-tom drum
`*Itrngl`	triangle; *triangle* (Fr.); *Triangel* (Ger.); *triangolo* (It.)
`*Ivibra`	vibraphone
`*Ixylo`	xylophone; *xylophone* (Fr.); *silofono* (It.)

1 s

1 l.

Keyboard Instruments

`*Iaccor`	accordion; *accordéon* (Fr.); *Akkordeon* (Ger.)
`*Icaril`	carillon
`*Icemba`	harpsichord; *clavecin* (Fr.); *Cembalo* (Ger.); *cembalo* (It.)
`*Iclavi`	clavichord; *clavicordium* (Lat.); *clavicorde* (Fr.)
`*Iclest`	celesta; *céleste* (Fr.)
`*Iforte`	fortepiano
`*Ihammd`	Hammond electronic organ
`*Iorgan`	pipe organ; *orgue* (Fr.); *Orgel* (Ger.); *organo* (It.); *organo* (Span.); *organum* (Lat.)
`*Ipiano`	pianoforte
`*Iporta`	portative organ
`*Ireedo`	reed organ
`*Irhode`	Fender-Rhodes electric piano

`*Isynth`	keyboard synthesizer

SEE ALSO

instrument class (3)

REPRESENTATION

instrument class — instrument class designation

DESCRIPTION

Instruments (or voices) may be represented according to the type or class of instrument or voice. There exist a wide variety of ways of classifying instrumental resources — each of which has advantages and disadvantages. The *instrument class* tandem interpretation echoes the most common distinctions made by practising musicians: voices, stringed instruments, woodwind instruments, brass instruments, keyboard instruments, and percussion instruments. The 'keyboard' class may be deemed redundant, since keyboard instruments may be variously classified as percussion instruments (e.g. piano), wind instruments (e.g. organ), or stringed instruments (e.g. harpsichord). Despite this overlap, the separate keyboard class has been retained since it is a natural way for musicians to refer to a group of instruments. The **instrument class** tandem interpretation can be used to identify explicitly the class of instrument employed.

SIGNIFIERS

The following table identifies six pre-defined instrument classes.

`*ICstr`	string instrument
`*ICww`	woodwind instrument
`*ICbras`	brass instrument
`*ICklav`	keyboard instrument
`*ICidio`	percussion instrument (idiophone)
`*ICvox`	voice

Summary of **instrument class** *Signifiers*

SEE ALSO

instrument (3)

REPRESENTATION

key — major/minor key designation

DESCRIPTION

For many tasks it is helpful to identify explicitly the prevailing key of a passage or work. The **key** tandem interpretation permits an explicit analytic judgement of key to be encoded in a Humdrum representation.

Key tandem interpretations consist of a single asterisk, followed by a single upper- or lower-case letter (A-G), followed by one or more sharps (#) or flats (-), followed by a colon character. Upper-case letters are designate major keys whereas lower-case letters designate minor keys. (No other modes can be encoded using this tandem interpretation for key indications.) By way of example, the following tandem interpretation specifies the key of F-sharp minor:

```
*f#:
```

Successive key interpretations supercede one other. That is, if a key of C major is indicated, followed some measures later by a key of G major tandem interpretation, then the preceding C major designation is considered to be entirely superceded.

The key tandem interpretation also permits the explicit encoding of undefined or unknown keys (`*?:`) and key-less or atonal passages (`*X:`).

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for keys.

A-G	major diatonic key signifiers
a-g	minor diatonic key signifiers
#	sharp key signifier
–	flat key signifier
X	atonal key signifier
?	unknown key signifier
:	end of key interpretation delimiter

Summary of **key** *Signifiers*

EXAMPLES

Several examples of key indications are given below:

`*C:`	key of C major
`*c:`	key of C minor
`*F#:`	key of F-sharp major
`*B--:`	key of B double-flat major
`*X:`	atonal passage
`*?:`	key unknown or undefined

Examples of **key** *Interpretations*

SEE ALSO

key signature (3)

REPRESENTATION

key signature — key signature designation

DESCRIPTION

Key signatures indicate the prevailing arrangement of global accidentals throughout a musical passage. Two forms of key signature interpretations can be distinguished: *pitch-class* signatures in which the accidentals modify all pitches of a given pitch class, and *pitch-height* signatures in which the accidentals modify only certain pitches of a specific pitch height.

Key signature interpretations consist of a single asterisk, followed by either a single upper- or lower-case letter 'K', followed by an open square bracket, followed by a list of pitches, followed by a closed square bracket. Pitches listed within the square brackets indicate the modified pitches or pitch-classes. The lower-case `*k` designates a (common) pitch-class key signature; The upper-case `*K` designates a (rare) pitch-height key signature;

In a pitch-class key signature, the pitch list specifies the recipe of sharps, flats, and/or naturals given in the key signature. Diatonic pitch are identified by lower-case pitch letter names. Each pitch is followed by one or more sharps, or flats, or a natural. For example, the key signature for three sharps (F,C,G) would be: `*k[f#c#g#]`. The order of the accidentals within the accidental list corresponds to the order in which they would be printed in a visual rendering of the score. Double- and triple- sharps and flats are represented by repetition of the octothrope (#) or minus sign (-). It is possible to mix sharps and flats within a single signature, to encode unconventional orderings, and to encode precautionary key signatures (such as those consisting only of naturals). It is not permitted to mix sharps/flats/naturals for a single pitch.

In very rare cases, key signatures modify only those pitches at a specific pitch height. For example, it may be that a composer wishes only some B's to be flat. These "pitch height" key signatures are designated by the upper-case key-letter 'K'. The corresponding pitch list uses `**pitch`-like representations to identify the modified pitches. For example, the following key signature:

```
*K[B3-C4#F4#B4nE5-]
```

specifies that B3 and E5 are lowered, and that C4 and F4 are raised. In addition, this key signature includes an explicit natural on B4 to remind readers that this pitch remains unaltered.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for key signatures.

k	pitch-class signifier
K	pitch-height signifier
a-g	pitch signifiers (pitch-class key signatures only)
A-G	pitch signifiers (pitch-height key signatures only)
0-9	octave indicators (pitch-height key signatures only)
#	sharp signifier
##	double sharp
–	flat signifier
n	natural signifier

Summary of **key signature** *Signifiers*

EXAMPLES

Several examples of key signatures are given below:

`*k[f#c#]`	regular key signature containing F-sharp and C-sharp
`*k[b-e-a-]`	regular key signature containing three flats
`*k[bnenan]`	precaution key signature using naturals only
`*k[]`	key signature containing no sharps or flats
`*k[b-e-f#]`	mixed key signature containing both sharps and flats
`*k[f##]`	key signature containing a single double sharp
`*k[c#f#]`	key signature encoding an unorthodox ordering of sharps
`*K[C#4B-4]`	pitch-height key signature identify C4 and B4 as modified
`*K[C#5B-5]`	pitch-height key signature identify C5 and B5 as modified

Examples of **key signature** *Interpretations*

SEE ALSO

key (3)

REPRESENTATION

language — language designation

DESCRIPTION

The **language** tandem interpretation permits the identification of the language for a given textual or phonetic Humdrum representation.

Language tandem interpretations consist of a single asterisk, followed by the upper-case letter `L`, followed by a language keyword designator. Several pre-defined language tandem interpretations are illustrated in below.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for language designations.

A-Z	upper-case Roman letters
a-z	lower-case Roman letters

Summary of **language** *Signifiers*

EXAMPLES

Examples of pre-defined language designations are given below:

Bengali	*LBengali
Croatian	*LHrvatski
Czech	*LCecha
Danish	*LDansk
Dutch	*LNederlands
English	*LEnglish
Finnish	*LSuomi
French	*LFrancais
German	*LDeutsch
Hindi	*LHindi
Italian	*LItaliano
Japanese	*LNihongo
Latin	*LLatin
Mandarin	*LPinying
Norwegian	*LNorsk

Polish	*LPolski
Portugese	*LPortugues
Romanian	*LRomana
Russian	*LRusski
Serbian	*LSrbski
Spanish	*LEspanol
Swahili	*LSwahili
Swedish	*LSvenska
Urdu	*LUrdu
Xhosa	*LIsixhosa
Zulu	*LIsizulu

SEE ALSO

****IPA** (2), ****text** (2)

REPRESENTATION

meter signatures — meter signature designation

DESCRIPTION

The **meter signature** tandem interpretation permits the encoding of meter signatures for a Humdrum representation.

Meter signature tandem interpretations consist of a single asterisk, followed by the upper-case letter `M`, followed by a meter indication. Meter indications consist of a top ("numerator") portion and a bottom ("denominator") portion. These portions are separated by a slash character (/). The numerator portion of the meter signature must be an integer value (greater than zero) — with no fractional part. The numerator may be split into two or more integers separated by the plus sign (+) in order to specify the grouping of beats within the measure. The denominator portion must be conform to `**recip` duration designations (8=eighth, 2.=dotted half, 0=breve, 6=eighth note triplet, etc.). Sample meter signatures are shown in the following table:

`*M2/4`	simple duple (quarter duration)
`*M3/2`	simple triple (half duration)
`*M4/0`	simple quadruple (breve duration)
`*M6/8`	compound duple (six-eight meter)
`*M2/4.`	compound duple (dotted quarter beat)
`*M9/16`	compound triple (nine-sixteen)
`*M12/4`	compound quadruple (twelve-four)
`*M4/2.`	compound quadruple (dotted half beat)
`*M5/4`	irregular quintuple (quarter duration)
`*M3+2/4`	irregular quintuple (three plus two beats)
`*M2+2+3/8`	irregular septuple (two plus two plus three beats)
`*M3+3+2/8`	irregular octuple (three plus three plus two)
`*M19/6`	nineteen eighth-duration triplets per measure
`*M21/8..`	twenty-one doubly-dotted eighths per measure
`*M?`	meter unknown
`*MX`	ametric passage (no meter)

Examples of **meter signature** *interpretations.*

Note that it is possible to represent *ametric* passages (*MX) and passages with *unknown* meters (*M?). These representations are useful, for example, when encoding Gregorian chant or African and other non-western rhythms.

Occasionally, musical scores will contain an alternating pair of meters (such as 3/4, 6/8, 3/4, 6/8, etc.). Such alternating meters are often represented in printed scores by a single meter signature — such as 3/4 (6/8). The meter signature tandem interpretation does not cater to such shorthands since the representation is intended to be *local* in its effect. This means that each change of meter must be labelled individually.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for meter signatures.

`0-9`	number signifiers
`.`	augmentation dot
`/`	numerator-denominator delimiter
`M`	meter signature keyword letter
`X`	ametric indicator
`?`	unknown meter indicator
`+`	grouping indicator (numerator only)

Summary of **meter signature** *Signifiers*

SEE ALSO

key signature (3), **metpos** (3), **metpos** (4), **timebase** (3), **timebase** (4)

REPRESENTATION

MIDI channel — MIDI channel designation

DESCRIPTION

The **MIDI channel** tandem interpretation permits the encoding of notated MIDI channel for a Humdrum representation.

MIDI channel tandem interpretations consist of a single asterisk, followed by the keyword `Ch`, followed by an interger indicating the channel number.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for MIDI channel indicators.

0-9	integer numbers representing the channel

Summary of **MIDI channel** *Signifiers*

EXAMPLES

Several examples of MIDI channel indications are given below:

`*Ch4`	MIDI channel 4
`*Ch1`	MIDI channel 1

Examples of **MIDI channel** *Interpretations*

SEE ALSO

****MIDI** (2), **midi** (4), **perform** (4), **smf** (4)

REPRESENTATION

overlay/underlay — overlay/underlay designation

DESCRIPTION

The **overlay/underlay** tandem interpretations are used to indicate whether the information encoded in a given spine is printed above (overlay) or below (underlay) a given staff.

Overlay and underlay tandem interpretations consist of a single asterisk, followed by either the keyword `ueber` or `unter` respectively. Following one of these keywords is a staff number — where staff number 1 corresponds to the first staff at the top of the system. There may follow optional positioning information; this information consists of a number preceded by a colon, and followed by a unit indication. The letter 'c' indicates centimeters, the letter 'p' indicates points, the letter 'i' indicates inches, and the percent sign indicates spacing as a percentage of the staff width.

EXAMPLES

Several examples of overlay and underlay indications are given below:

`*ueber3:1i`	overlay positioned 1 inch above the center of the third staff in the system
`*ueber8:3.4c`	overlay positioned 3.4 cm above the center of the eighths staff in the system
`*unter1:82p`	overlay positioned 82 points above the center of the first staff in the system
`*unter2:50%`	overlay positioned 50 percent of the staff width above the center of the first staff in the system

Examples of **overlay/underlay** *Tandem Interpretations*

SEE ALSO

staff (3), **staff lining** (3), ****text** (2)

REPRESENTATION

section labels — section label designations

DESCRIPTION

Section labels are tandem interpretation that are used to identify segments or sections of some Humdrum representation. Section labels are useful for identifying logical divisions or passages, such as expositions, codas, second endings, rehearsal segments, etc. Section labels provide useful markers for extracting passages using the Humdrum **yank** command. Section labels are also used in conjunction with Humdrum **expansion lists** to permit the encoding of "abbreviated format" files. (See the **thru (4)** command.)

Section labels consist of a single asterisk, followed by the greater-than sign (>), followed by a keyword (or label) that names the section. Note that labels may contain spaces, hence `*>1st ending` is a legitimate section label. In abbreviated format files, each section must be designated by a unique name.

Humdrum sections formally begin with a section label. Sections end when either another section label is encountered, or when all spines are assigned new exclusive interpretations, or when all spines terminate. Sections cannot be nested. Whenever a section label is encoded, the identical label must be repeated across all concurrent spines. That is, all tokens in any given data record must belong to the same section — without regard for the spines.

SIGNIFIERS

Section labels 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.

EXAMPLES

Several examples of section labels indications are given below:

```
*>CODA
*>refrain
*>Dal Segno
*>Verse #3
*>Rehearsal Marking J
*>E
```

Examples of **section label** *Interpretations*

SEE ALSO

expansion lists (3), **thru** (3), **thru** (4), **yank** (4)

REPRESENTATION

spine paths — spine path indicators

DESCRIPTION

Spine path indicators are special types of tandem interpretations that permit the encoding of potentially complex spine-path changes in a Humdrum representation.

Humdrum spines may be terminated, added, split, joined, or exchanged. Spine-path indicators consist of a single asterisk, followed by one of five key-letters: the plus sign, the minus sign, the carret, the lower-case letter 'v', or the lower-case letter 'x'.

Note that spine-path indicators cannot appear on the same line with non-spine-path interpretations — such as key signatures or clefs. That is, spine-path indicators cannot be mixed with other tandem or exclusive interpretations on the same records. Several spine-path indicators may share the same record, however.

SIGNIFIERS

The following table identifies all five spine path indicators.

`*+`	add 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

Summary of **spine paths** *Signifiers*

EXAMPLES

A number of examples of the use of spine path indicators are given in the discussion entitled "Spine Paths" in Section 1 of this manual.

SEE ALSO

humdrum (4)

REPRESENTATION

staff — staff designation

DESCRIPTION

The **staff** tandem interpretation can be used to assign a given spine to a particular staff within a system.

Staff tandem interpretations consist of a single asterisk, followed by the keyword `staff`, followed by an integer identifying a staff, where `staff1` corresponds to the first staff at the top of the system. See EXAMPLES below.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for staff.

staff	staff lining keyword
0-9	numbers

Summary of **staff** *Signifiers*

EXAMPLES

Several examples of staff indications are given below:

`*staff1`	top-most staff in the system
`*staff12`	twelfth staff from the top of the system
`*staff0`	no meaning

Examples of **staff** *Interpretations*

SEE ALSO

staff lining (3)

REPRESENTATION

staff lining — staff lining designation

DESCRIPTION

The **staff lining** tandem interpretation permits the detailed encoding of the number of lines in a staff, and also whether individual lines are dotted, colored, or invisible.

Staff lining tandem interpretations consist of a single asterisk, followed by a keyword consisting of the vertical bar followed by a period (i.e. '*|.'), followed by one or more signifiers characterizing successive lines beginning at the bottom of the staff. See EXAMPLES below.

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for staff lining.

|.	staff lining keyword
|	black line indicator
:	dotted line indicator
R	*ruber*, red line
V	*viridis*, green line
C	*caeruleus*, blue line
X	invisible line
0	(zero) no staff lines indicator

Summary of **staff lining** *Signifiers*

EXAMPLES

Several examples of staff lining indications are given below:

`*|.|||||`	five-line staff
`*|.||||`	four-line staff
`*|.|`	single-line staff
`*|.0`	no staff lines
`*|.|X|`	three-line staff with middle line invisible
`*|.||R|`	four-line staff with the third line colored red

Examples of **staff lining** *Interpretations*

SEE ALSO

clef (3), **staff** (3)

REPRESENTATION

strophe — strophic passage designators

DESCRIPTION

The **strophe** tandem interpretations are used to encode alternative parallel paths of sequential information. Strophic representations are useful for such tasks as representing texts for different verses of a song, or for indicating alternative interpretations of a sequence of notes — such as *ossia* passages.

Strophic passages begin from a single spine that splits into several "alternative" spines — which later rejoin to form a single spine again. Four different tandem interpretations are involved in the encoding of strophic passages. These include the *strophic passage initiator,* the *strophic passage terminator,* the *strophe labels,* and the *strophe end indicators.*

Each strophic passage begins with a *strophic passage initiator.* This consists simply of a single asterisk followed by the keyword "strophe" (i.e.`*strophe`). This tandem interpretation marks a single spine that is about to be split into alternative parallel paths. When the alternative spines are ultimately rejoined, a *strophic passage terminator* marks the end of the strophic passage. This terminator consists simply of a single asterisk followed by the upper-case letter 'S', followed by a minus sign (i.e.`*S-`).

Following a strophic passage initiator, the spine is split into the required number of alternative spines using the Humdrum split interpretation (see spine paths). Each spine is then identified using a strophe label. Strophe labels are tandem interpretations that begin with a single asterisk, followed by the upper-case letter 'S', followed by a slash (/), followed by a unique name consisting of numbers and/or alphabetic letters. For example, where a composer has notated an alternative way of performing a passage ("*ossia*"), two strophe labels may be defined:

```
*S/sic   *S/ossia
```

If the strophic data imply some sort of order (such as verses in a song), simple numerical labels should be used:

```
*S/1   *S/2   *S/3   *S/4
```

The use of numbers is important when the file is expanded using the **thru** and **strophe** commands.

Following the strophic data records, each strophic spine is ended using the strophe end indicator. This tandem interpretation consists of an asterisk, followed by the upper-case letter 'S', followed by a slash, followed by the keyword 'fin' (i.e.`*S/fin`).

SIGNIFIERS

The following table summarizes the four types of tandem interpretations used for strophic passages.

`*strophe`	strophic passage initiator
`*S/`*n.n*	numerical strophe label
`*S/`*name*	named strophe label
`*S/fin`	strophe end indicator
`*S-`	strophic passage terminator

Summary of **strophe** *Tandem Interpretations*

EXAMPLES

For examples of strophic passages, see the **strophe** command description in Section 4 of this manual.

SEE ALSO

expansion lists (3), **strophe** (4)

REPRESENTATION

tempo — tempo designation

DESCRIPTION

The **tempo** tandem interpretation permits the encoding of gross overall tempo for a Humdrum representation.

Tempo tandem interpretations consist of a single asterisk, followed by the keyword `MM`, followed by one of four possible types of tempo indications. Tempo indications may consist of a single real or integer value specifying the the number of quarter-durations per minute, such as `*MM96`. Alternatively, tempo *ranges* may be encoded by interposing a hyphen between two numerical values, such as `*MM55.5-56.3`. Once again, the numerical values pertain to the number of quarter-durations per minute. Instead of numerical specifications, conventional Italian tempo terms may be encoded in square brackets, such as `*MM[Presto]`. Finally, an "unknown tempo" can be explicitly represented by the presence of a question mark, i.e. `*MM?`. (See EXAMPLES below.)

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for tempo.

`MM`	tempo keyword
`0-9`	numbers
`.`	decimal point
`-`	range signifier
`[ ]`	tempo-term delineators
`A-Z`	tempo-term characters
`a-z`	tempo-term characters
<*space*>	tempo-term space

Summary of **tempo** *Signifiers*

EXAMPLES

Several examples of tempo indications are given below:

`*MM60`	tempo of 60 quarter-durations per minute
`*MM60.`	tempo of 60 quarter-durations per minute
`*MM60.0`	tempo of 60 quarter-durations per minute
`*MM96.3`	tempo of 96.3 quarter-durations per minute
`*MM72-78`	tempo range between 72 and 78 quarter-durations per minute
`*MM51.2-51.4`	tempo range between 51.2 and 51.4 quarter-durations per minute
`*MM[Largo]`	tempo Largo
`*MM[Allegro molto]`	Allegro molto tempo
`*MM?`	unknown tempo

Examples of **tempo** *Interpretations*

SEE ALSO

midi (4), **perform** (4)

REPRESENTATION

thru — through-composed format designation

DESCRIPTION

The **thru** tandem interpretation identifies a given Humdrum representation as being in a *through-composed format.*

Musical scores are often notated to take advantage of repetitions in the music. Notational devices such as repeat marks and Da Capos can be encoded using Humdrum *section labels* and *expansion lists;* the resulting succinct representations are called *abbreviated format* files.

Abbreviated formats are implicitly indicated by the presence of an *expansion list* encoded prior to any data records. When an abbreviated format file is expanded using the **thru** command, any expansion lists present in the input are discarded. The presence of a `*thru` tandem interpretation in a file (prior to any data) explicitly identifies the file as being in a through-composed format rather than abbreviated format.

SIGNIFIERS

The `*thru` tandem interpretations consist simply of a single asterisk, followed by the keyword `thru`.

SEE ALSO

expansion lists (3), **section labels** (3), **thru** (4)

REPRESENTATION

timebase — timebase designation

DESCRIPTION

The **timebase** tandem interpretation permits the encoding of notated timebase for a Humdrum representation.

Timebase tandem interpretations consist of a single asterisk, followed by the keyword `tb`, followed by a `**recip`-like encoded duration. Durations consist of a single integer followed by zero or more periods (representing augmentation dots).

SIGNIFIERS

The following table summarizes the mappings of signifiers and signifieds for timebase.

tb	timebase keyword
0-9	`**recip`-like durations
.	augmentation dot

Summary of **timebase** *Signifiers*

EXAMPLES

Several examples of timebase indications are given below:

`*tb32`	thirty-second note timebase
`*tb8.`	dotted eighth note timebase

Examples of **timebase** *Interpretations*

SEE ALSO

assemble (4)

REPRESENTATION

transposition — transposition designation

DESCRIPTION

The **transposition** tandem interpretation permits the encoding of transposition information for a given pitch-related spine.

Transpositions can be characterized by the diatonic letter name shift, and the chromatic semitone shift. For example, transposing a pitch from C to D may be regarded as a diatonic letter name shift of up 1, as well as a chromatic semitone shift up 2. By contrast, a transposition from C to C double-sharp may be characterized as a diatonic letter name shift of 0, as well as a chromatic semitone shift up 2.

Transposition tandem interpretations consist of a single asterisk, followed by either the keyword `Tr` or `ITr`, followed by the lower-case letter 'd', followed by a signed integer, followed by the lower-case letter 'c', followed by another signed integer. The first integer indicates the diatonic letter name shift, while the second integer indicates the chromatic semitone shift. The keyword `Tr` indicates a transposition; the keyword `ITr` indicates a transposing instrument.

EXAMPLES

Several examples of transposition interpretations are given below:

`*Trd1c2`	encoded data is transposed up 1 diatonic letter name; up 2 semitones
`*Trd-1c-2`	encoded data is transposed down 1 diatonic letter name; down 2 semitones
`*Trd0c1`	encoded data is transposed up 1 semitones; same diatonic letter names
`*Trd0c-1`	encoded data is transposed down 1 semitones; same diatonic letter names
`*ITrd-1c-2`	transposing instrument; encoded data is at concert pitch; original score was *notated* up 1 diatonic letter name; up 2 semitones

Examples of **transposition** *Interpretations*

SEE ALSO

****kern** (2), **key** (3), **key signature** (3), **trans** (4),

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      **selfg   **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
*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 0" 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)

Section 5

Regular Expression Reference

Regular Expression Documentation

This section of the *Reference Manual* describes in detail the operation of regular expressions. Regular expressions were developed on UNIX operating systems as a generic way of defining patterns of characters. The following documentation consists of two parts: (1) a tutorial introduction to regular expressions, and (2) a comprehensive summary of the syntax.

Regular Expressions: A Tutorial Introduction

A common task in computing environments is searching through some set of data for occurrences of a given pattern. When a pattern is found, various courses of action may be taken. The pattern may be copied, counted, deleted, replaced, isolated, modified, or expanded. A successful pattern-match might even be used to initiate further pattern searches.

Regular expression syntax provides a standardized method for defining patterns of characters. Regular expressions are restricted to common ASCII characters — including the upper- and lower-case letters of the alphabet, the digits zero to nine, and other special characters typically found on typewriter-like keyboards.† Since all Humdrum representations are based on strings of ASCII characters, regular expressions can be used to identify patterns of musical signifiers. Regular expressions can be used with any type of Humdrum representation; it does not matter what type of information is represented by the signifiers.

A number of general-purpose commands rely on regular expression syntax for specifying patterns. These include the UNIX **awk, ed, egrep, ex, expr, grep, gres, pg, sed** and **vi** commands. In addition, more than a dozen Humdrum tools rely on regular expression syntax for specifying patterns of ASCII characters. Included in these tools are the Humdrum **correl**, **fill**, **mint**, **num**, **patt**, **pattern**, **recode**, **rend**, **xdelta** and **yank** commands.

For musicologist-users interested in searching for complex music-related patterns, it is valuable to develop some facility in using regular expressions. Regular expressions will not allow users to define every possible musical pattern of potential interest. In particular, regular expressions cannot be used to identify deep-structure patterns from surface-level representations. However, regular expressions are quite powerful — typically more powerful than they appear to the novice user. Not all users will be equally adept at formulating an appropriate regular expression to search for a given pattern. As with the study of a musical instrument, practise is advised.

Literals

The simplest regular expressions are merely literal sequences of characters forming a character **string,** as in the pattern:

```
foo
```

This pattern will match any data string containing the sequence of letters f-o-o. The letters must be contiguous, so no character (including spaces) can be interposed between any of the letters. The above pattern is present in strings such as "`fool`" and "`mgfooXy`" but not in strings such as "`Foo`" or "`follow`". The above pattern is called a *literal* since the matching pattern must be literally identical to the regular expression (including the correct use of upper- or lower-case).

† The initialism 'ASCII' stands for the American Standard Code for Information Interchange.

When a pattern is found, a starting point and an ending point are identified in the input string, corresponding to the defined regular expression. The specific sequence of characters found in the input string is referred to as the *matched string* or *matched pattern.*

Wild-Card

Patterns that are not literal include so-called *metacharacters.* Metacharacters are used to specify various operations, and so are not interpreted as their literal selves. The simplest regular expression metacharacter is the period (`.`). The period matches *any single character* — including spaces, tabs, and other ASCII characters. For example, the pattern:

```
f.o
```

will match any input string containing three characters, the first of which is the lower-case '`f`' and the third of which is the lower-case '`o`'. Hence, the above pattern is present in strings such as "`flow`" and "`proof of`" but not in "`follow`" or "`found`". Any character can be interposed between the '`f`' and the '`o`' provided there is precisely just one such character.

Escape Character

A problem with metacharacters such as the period is that sometimes the user wants to use them as literals. The special meaning of metacharacters can be "turned-off" by preceding the metacharacter with the backslash character (\). The backslash is said to be an *escape* character since it is used to release the metacharacter from its normal function. For example, the regular expression:

```
\.
```

will match the period character. The backslash itself may be escaped by preceding it by an additional backslash (i.e. \\).

Repetition Operators

Another metacharacter is the plus sign (+). The plus sign means "one or more consecutive instances of the previous expression." For example,

```
fo+
```

specifies any character string beginning with a lower-case '`f`' followed by one or more consecutive instances of the small letter '`o`'. This pattern is present in such strings as "`food`" and "`folly`," but not in "`front`" or "`flood`." Notice that the length of the matched string is changeable. In the case of "`food`" the matched string consists of three characters, whereas in "`folly`" the matched string consists of just two characters. Notice also that the plus sign modifies only the preceding letter '`o`' — that is, the single letter '`o`' is deemed to be the *previous expression* which is affected by the +. However, the affected expression need not consist of just a single character. In regular expressions, parentheses ( ) are metacharacters that can be used to bind several characters into a single unit or sub-expression. Consider, by way of example, the following regular expression:

```
(fo)+
```

The parentheses now bind the letters 'f' and 'o' into a single expression, and it is this expression that is now modified by the plus sign. The above regular expression may be read as "one or more consecutive instances of the string 'fo'." This pattern is present in strings like "food" (one instance) and "fofoe" (two instances).

Several sub-expressions may occur within a single regular expression. For example, the following regular expression means "one or more instances of the letter 'a', followed by one or more instances of the string 'bl'."

```
(a)+(bl)+
```

This would match character strings in inputs such as "able" and "kraable," but not in "dabble" (two consecutive b's) or "blbl" (no leading 'a'). Note that the parentheses are not required around the letter 'a' in the above regular expression. The following expression mixes the + repetition operator with the wild-card (.):

```
c+.m+
```

This pattern is present in strings such as "accompany," "accommodate," and "cymbal." This pattern will also match strings such as "ccm" since the second 'c' can be understood to match the period character.

A second repetion operator is the asterisk (*). The asterisk means "zero or more consecutive instances of the previous expression." For example,

```
fo*r
```

specifies any character string beginning with a lower-case 'f' followed by zero or more instances of the letter 'o' followed by the letter 'r'. This pattern is present in such strings as "ford," "foooorm" as well as "fred," and "front." As in the case of the plus sign, the asterisk modifies only the preceding expression — in this case the letter 'o'. Multi-character expressions may be modified by the asterisk repetition operator by placing the expression in parentheses. Thus, the regular expression:

```
ba(na)*
```

will match strings such as "ba," "bana," "banana," "bananana," etc.

Incidentally, notice that the asterisk metacharacter can be used to replace the plus sign (+) metacharacter. For example, the regular expression X+ is the same as XX*. Similarly, (abc)+ is equivalent to (abc)(abc)*. The plus sign (+) metacharacter is not strictly necessary, but it is frequently more convenient.

A frequent construct used in regular expressions joins the wild-card (`.`) with the asterisk repetition character (`*`). The regular expression:

```
.*
```

means "zero or more instances of any characters." (Notice the plural "characters;" this means the repetition need not be of one specific character.) This expression will match any string, including nothing at all (the *null string*). By itself, this expression is not very useful. However it proves invaluable in combination with other expressions. For example, the expression:

```
{.*}
```

will match any string beginning with a left curly brace and ending with a right curly brace. If we replaced the curly braces by the space character, then the resulting regular expression would match any string of characters separated by spaces — such as printed words.

A third repetition operator is the question mark (`?`) — which means "zero or one instance of the preceding expression." This metacharacter is frequently useful when you want to specify the presence or absence of a single expression. For example,

```
fl?o
```

will match "`flow`" and "`fodder`" but not "`fly`" or "`fllo`."

Once again, parentheses can be used to specify more complex expressions. The pattern:

```
fl?(o)+
```

is present in such strings as "`flow`," "`food`," and "`flood`," but not in "`fllox`" or "`frown`."

In summary, we've identified three metacharacters pertaining to the *number of occurrences* of some character or string. The plus sign means "one or more," the asterisk means "zero or more," and the question mark means "zero or one."† Collectively, these metacharacters are know as *repetition operators* since they indicate the number of times an expression can occur in order to match.

Min-Max Character Repetition

In addition to these general repetition operators, there is a syntax to specify the precise number of occurrences for a single character, or a numerical range of possible repetitions. Three syntactical forms are provided. All three forms use the special delimiters \{ and \}. The first form specifies the exact number of repetitions:

† Sometimes it is difficult to remember which metacharacter has which effect. The follow mnemonic may help: question mark connotes "it's either there or it's not;" the plus sign connotes "maybe there's an addition one;" and the asterisk connotes "match the universe — including nothing."

```
x\{10\}
```

This regular expression will match precisely 10 consecutive occurrences of the lower-case letter 'x'. The second form specifies the minimum number of repetitions:

```
x\{10,\}
```

(Note the presence of the comma.) This regular expression will match an entire string of 'x's that consists of *at least* 10 consecutive occurrences of 'x'. The third form specifies a minimum and maximum number of repetitions:

```
x\{10,20\}
```

This regular expression will match a string of 'x's that consists of at least 10 occurrences, but not more than 20 occurrences of 'x'.

Priority of the Longest String Match

When matching regular expressions to some input, the operation procedes from left to right, and *longer matching strings take priority over shorter matching strings.* Consider, for example, an input record (line) consisting of 29 consecutive 'x's. Given the regular expression used above,

```
x\{10,20\}
```

the input would be parsed as follows. The first 10 'x's would satisfy the minimum length criterion and so, by themselves, would constitute a "matching string." In encountering the eleventh letter 'x', this longer string would be recognized as also satisfying the regular expression. Hence, the end-point of the matching string would be moved to the eleventh 'x' and the previous 10-character matching-string would be superceded. A similar process will continue until the twentieth consecutive letter 'x' is encountered. With the 21st 'x', the maximum length criterion prevents this 'x' from joining the preceding 20 as a matching string. Having matched this maximum length string, the regular expression parser will continue from the 21st letter 'x' to see if another matching string can be found. Since there are only nine remaining 'x's, no further matching strings exist in this input. Although it is possible to conceive of 29 'x's as two matching strings consisting of (say) 19 and 10 consecutive 'x's respectively, in this case the regular expression parser will identify only a single matching string. Once again, the longest matching string takes precedence over shorter potential matching strings.

Context Anchors

Often it is helpful to limit the number of occurrences matched by a given pattern. You may want to match patterns in a more restricted context. One way of restricting regular expression pattern-matches is by using so-called *anchors.* There are two regular expression anchors. The caret (^) anchors the expression to the beginning of the string. The dollar sign ($) anchors the expression to the end of the string. For example,

```
^A
```

matches the upper-case letter 'A' only if it occurs at the beginning of a string. Conversely,

```
A$
```

will match the upper-case letter 'A' only if it is the last character in a string. Both anchors may be used together, hence the following regular expression matches only those strings containing just the letter 'A':

```
^A$
```

Of course anchors can be used in conjunction with the other regular expressions we have seen. For example, the regular expression:

```
^a.*z$
```

matches any string that begins with 'a' and ends with 'z'.

OR Logical Operator

One of several possibilities may be matched by making use of the logical *OR* operator — represented by the vertical bar (|) metacharacter. For example, the following regular expression matches either the letter 'x' or the letter 'y' or the letter 'z':

```
x|y|z
```

Expressions may consist of multiple characters, as in the following expression which matches the string 'sharp' or 'flat' or 'natural'.

```
sharp|flat|natural
```

More complicated expressions may be created by using parentheses. For example, the regular expression:

```
(simple|compound) (duple|triple|quadruple|irregular) meter
```

will match eight different strings, including `simple triple meter` and `compound quadruple meter`.

Character Classes

In the case of single characters, a convenient way of identifying or listing a set of possibilities is to use the regular expression *character class*. For example, rather than writing the expression:

```
a|b|c|d|e|f|g
```

the expression may be simplified to:

```
[abcdefg]
```

Any character within the square brackets (a "character class") will match. Spaces, tabs, and other characters can be included within the class. When metacharacters like the period (`.`), the asterisk (`*`), the plus sign (+), and the dollar sign (`$`) appear in character classes, they lose their special meaning, and become simple literals. Thus the regular expression:

```
[xyz.+*$]
```

matches any of the characters '`x`,' '`y`,' '`z`,' the period, plus sign, asterisk, or the dollar sign.

Some other characters take on special meanings within character classes. One of these is the dash (–). The dash sign acts as a *range* operator. For example,

```
[A-Z]
```

represents the class of all upper-case letters from A to Z. Similarly,

```
[0-9]
```

represents the class of digits from zero to nine. The expression given earlier — `[abcdefg]` — can be simplified further to: `[a-g]`. Several ranges can be mixed within a single character class:

```
[a-gA-G0-9#]
```

This regular expression matches any of the lower- and upper-case characters from A to G, or any digit, or the octothorpe (#). If the dash character appears at the beginning or end of the character class, it loses its special meaning and become a literal dash, as:

```
[a-gA-G0-9#-]
```

This regular expression adds the dash character to the list of possible matching characters.

Another useful metacharacter within character classes is the caret (^). When the caret appears at the beginning of a character-class list, it signifies a *complementary character class.* That is, only those characters *not* in the list are matched. For example,

```
[^0-9]
```

matches any character other than a digit. If the caret appears in any position other than at the beginning of the character class, it loses its special meaning and is treated as a literal. Note that if a character-class range is not specified in numerically ascending order or alphabetic order, the regular expression is considered ungrammatical and will result in an error.

Character Class Keywords

Some character class expressions occur frequently in pattern definitions. The ten most common character class expressions can be invoked via character-class *keywords*. These are listed below:

[[:upper:]]	match any upper-case alphabetic character (same as [A-Z])
[[:lower:]]	match any lower-case alphabetic character (same as [a-z])
[[:alpha:]]	match any upper- or lower-case alphabetic character (same as [a-zA-z])
[[:digit:]]	match any single numerical digit (same as [0-9])
[[:alnum:]]	match any alphanumeric character (same as [0-9a-zA-Z])
[[:space:]]	match any empty space (blank, tab)
[[:graph:]]	match any grapheme (any character except empty space)
[[:print:]]	match any printable character (empty space included)
[[:punct:]]	match any non-alphanumeric character
[[:cntrl:]]	match any non-printable character, including control characters

Character-class keywords for regular expressions

Multiple character-class keywords can appear together, hence

```
[[:lower:][:digit:]]
```

with match any lower-case alphabetic character, or any digit.

Examples of Regular Expressions

The following table lists some examples of regular expressions and provides a summary description of the effect of each expression:

<table>
<tr><td>A</td><td>match letter ‘A’</td></tr>
<tr><td>^A</td><td>match letter ‘A’ at the beginning of a string</td></tr>
<tr><td>A$</td><td>match letter ‘A’ at the end of a string</td></tr>
<tr><td>.</td><td>match any character (including space or tab)</td></tr>
<tr><td>A+</td><td>match one or more instances of letter ‘A’</td></tr>
<tr><td>A?</td><td>match a single instance of ‘A’ or the null string</td></tr>
<tr><td>A*</td><td>match one or more instances of ‘A’ or the null string</td></tr>
<tr><td>.*</td><td>match any string, including the null string</td></tr>
<tr><td>A.*B</td><td>match any string starting with ‘A’ up to and including ‘B’</td></tr>
<tr><td>A|B</td><td>match ‘A’ or ‘B’</td></tr>
<tr><td>(A)|(B)</td><td>match ‘A’ or ‘B’</td></tr>
<tr><td>[AB]</td><td>match ‘A’ or ‘B’</td></tr>
<tr><td>[^AB]</td><td>match any character other than ‘A’ or ‘B’</td></tr>
<tr><td>AB</td><td>match ‘A’ followed by ‘B’</td></tr>
<tr><td>AB+</td><td>match ‘A’ followed by one or more ‘B’s</td></tr>
<tr><td>(AB)+</td><td>match one or more instances of ‘AB’, e.g. ABAB</td></tr>
<tr><td>(AB)|(BA)</td><td>match ‘AB’ or ‘BA’</td></tr>
<tr><td>A\{5\}</td><td>match five instances of ‘A’</td></tr>
<tr><td>A\{5,\}</td><td>match five or more instances of ‘A’</td></tr>
<tr><td>A\{5,9\}</td><td>match between five and nine instances of ‘A’</td></tr>
<tr><td>[^A]AA[^A]</td><td>match two ‘A’s preceded and followed by characters other than ‘A’s</td></tr>
<tr><td>^[^^]</td><td>match any character at the beginning of a record except the caret</td></tr>
</table>

Examples of regular expressions.

Examples of Regular Expressions in Humdrum

The following table provides some examples of regular expressions pertinent to Humdrum-format inputs:

<table>
<tr><td>^!!</td><td>match any global comment</td></tr>
<tr><td>^!!.*Beethoven</td><td>match any global comment containing ‘Beethoven’</td></tr>
<tr><td>^!!.*[Rr]ecapitulation</td><td>match any global comment containing the word
‘Recapitulation’ or ‘recapitulation’</td></tr>
<tr><td>^!($|[^!])</td><td>match only local comments</td></tr>
<tr><td>^**</td><td>match any exclusive interpretation</td></tr>
<tr><td>^*[^*]</td><td>match only tandem interpretations</td></tr>
<tr><td>^*[-+vx^]$</td><td>match spine-path indicators</td></tr>
<tr><td>^[^*!]</td><td>match only data records</td></tr>
<tr><td>^[^*!].*$</td><td>match entire data records</td></tr>
<tr><td>^(\.<<i>tab</i>>)*\.$</td><td>match records containing only null tokens (<i>tab</i> means a tab)</td></tr>
<tr><td>^*f#:</td><td>match key interpretation indicating F# minor</td></tr>
</table>

Regular expressions suitable for all Humdrum inputs.

By way of illustration, the next table shows examples of regular expressions appropriate for processing `**kern` representations.

<table>
<tr><td>^=</td><td>match any **kern barline or double barline</td></tr>
<tr><td>^=[^=]</td><td>match **kern single barlines but not double barlines</td></tr>
<tr><td>^[^=]</td><td>match any token other than a barline or double barline</td></tr>
<tr><td>;</td><td>match any **kern note or barline containing a pause</td></tr>
<tr><td>T</td><td>match any **kern note containing a whole-tone trill</td></tr>
<tr><td>[Tt]</td><td>match any **kern note containing a whole-tone or half-tone trill</td></tr>
<tr><td>-</td><td>match any **kern note containing at least one flat</td></tr>
<tr><td>[#]</td><td>match any **kern note containing a sharp, double-sharp, etc.</td></tr>
<tr><td>[#n-]</td><td>match any **kern note containing an accidental, including a natural</td></tr>
<tr><td>[A-Ga-g]+</td><td>match any diatonic pitch letter-name</td></tr>
<tr><td>[0-9]+\.</td><td>match **kern dotted durations</td></tr>
<tr><td>[0-9]+\.\.[^.]</td><td>match only doubly-dotted durations</td></tr>
<tr><td>[Gg]+[^#-]</td><td>match any **kern pitch ‘G’ that does not have a sharp or flat</td></tr>
<tr><td>(^|[^g])gg($|[^g#-])</td><td>match only the pitch ‘gg’ (G5)</td></tr>
<tr><td>{.*r|r.*{</td><td>match all phrase beginnings that start with a rest</td></tr>
<tr><td>^4[^0-9.]|[^0-9]4([^0-9.]|$)</td><td>match **kern quarter durations</td></tr>
<tr><td>^(8|16)[^0-9.]|[^0-9](8|16)[^0-9.]</td><td>match eighth and sixteenth durations only</td></tr>
<tr><td>(([Ee]+-)|([Gg]+-)|([Bb]+-))($|[^-])</td><td>match any note from E-flat minor chord</td></tr>
</table>

Regular expressions suitable for `**kern` *data records.*

Note that the above regular expressions assume that comments and interpretations are not processed in the input. The processing of just data records can be assured by embedding each of the regular expressions given above in the expression

`(^[^*!].*`*regexp*`)|(^`*regexp*`)`

For example, the following regular expression can be used to match **kern trills without possibly mistaking comments or interpretations:

```
(^[^*!].*[Tt])|(^[Tt])
```

For Humdrum commands such as **humsed, rend, xdelta, yank,** and *ydelta,* regular expressions are applied only to data records so there is no need to use the more complex expressions. In some circumstances, the **rid** command might be used to explicitly remove comments and interpretations prior to processing.

Basic, Extended, and Humdrum-Extended Regular Expressions

Over the years, new features have been added to regular expression syntax. Some of the early software tools that make use of regular expressions do not support the extended features provided by more recently developed tools. So-called “basic” regular expressions include the following features: the single-character wild-card (`.`), the repetition operators (`*`) and (`?`) — but not (+), the context anchors (`^`) and (`$`), character classes (`[...]`), and complementary character classes (`[^...]`). Parenthesis-grouping is supported in Basic regular expressions, but the parentheses must be used in conjunction with the backslash to *enable* this function (i.e. `\(  \)`).

So-called "extended" regular expressions include the following (added features are highlighted in bold): the single-character wild-card (`.`), the repetition operators (`*`), (**?**) and (**+**), **min-max character repetition** (`\{  \}`), the context anchors (`^`) and (`$`), character classes (`[...]`), complementary character classes (`[^...]`), **character-class keywords** (`[:...:]`), **the logical OR** (`|`), and parenthesis-grouping.

Record-Repetition Operators

The Humdrum **pattern** command permits an additional regular expression feature that is especially useful in musical applications. Specifically, **pattern** permits the defining of patterns spanning more than one line or record. Record-repetition operators are specified by following the regular expression with a tab — followed by either `+`, `*`, or `?`. For example, consider the following Humdrum-extension regular expression:

```
X	+
Y	*
Z	?
```

When the metacharacters `+`, `*`, or `?` appear at the end of a record, preceded by a tab character, they pertain to the number of records, rather than the number of repetitions of the expression within a record. The first record of the regular expression (`X`<*tab*>`+`) will match one or more successive lines each containing the letter '`X`'. The second record of the regular expression (`Y`<*tab*>`*`) will match zero or more subsequent lines containing the letter '`Y`'. The third record of the regular expression (`Z`<*tab*>`?`) will match zero or one line containing the letter '`Z`'. Hence, the above multi-record regular expression 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 regular expression would match an input containing one line containing the letter '`X`'.

Record-repetition operators can be used in conjunction with all of the other regular expression features. For example, the following regular expression matches one or more successive `**kern` data records containing the pitch '`G`' (naturals only) followed optionally by a single '`G#`' followed by one or more records containing one or more pitches from an A major triad — the last of which must end a phrase:

```
[Gg]+[^#-]	+
[Gg]+#[^#]	?
([Aa]+|([Cc]+#)|[Ee]+)[^#-]		*
(}.*([Aa]+|([Cc]+#)|[Ee]+)[^#-]))|(([Aa]+|([Cc]+#)|[Ee]+)[^#-].*})
```

NAME

regexp — regular expression pattern-match syntax

DESCRIPTION

"Regular expression syntax" provides a standardized way of defining pattern of characters. Regular expressions are limited to common ASCII characters — such as the letters of the alphabet, numbers, and other special characters typically found on typewriter-like keyboards.

Three variants of regular expression syntax can be distinguished: (1) basic regular expressions, (2) extended regular expressions, and (3) Humdrum-extended regular expressions. The differences are outlined later in this documentation.

SYNTAX

A regular expression is any combination of sub-expressions consisting of literals, wild cards, repetition operators, min-max character repetition, context anchors, character classes, complementary character classes, the logical OR, and parenthesis-grouping.

Literals. A literal is any string not containing unescaped metacharacters. Metacharacter may be treated as literal characters by preceding the metacharacter by the escape character — the backslash (\). (The backslash itself may be escaped by preceding it by an additional backslash.) Literals are matched only if a string of characters is found that is identical to the literal.

Wild Card (.) The period character (`.`) is a wild-card that matches any single character.

Repetition Operators (+) (?) (*). An *expression* followed by the plus sign (+) matches any string of one or more occurrences of *expression*. An *expression* followed by the question mark (`?`) matches zero or one occurrence of *expression*. An *expression* followed by the asterisk (`*`) matches zero or more occurrences of *expression*.

Min-Max Character Repetition (\{ \}). Precise numbers of occurrences for a single character, or minimum and maximum numbers of occurrences can be specified using the special delimiters \{ and \}. Three syntactical forms exist; the regular expression `x\{10\}` will match precisely 10 occurrences of the lower-case letter 'x'. The regular expression `x\{10,\}` will match an entire string of 'x's that consists of *at least* 10 occurrences of 'x'. The regular expression `x\{10,20\}` will match a string of 'x's that consists of at least 10 occurrences, but not more than 20 occurrences of 'x'.

Context Anchors (^) ($). An *expression* preceded by the caret anchor (`^`) matches only those occurrences of *expression* starting at the beginning of a string. An *expression* followed by the dollar sign anchor (`$`) matches only those occurrences of *expression* ending at the end of a line.

Character Classes ([...]). Any character given in a string bounded by left '[' and right ']' braces will be matched. Character ranges can be indicated via the dash sign (-), hence the character class [A-Z] matches any upper-case letter, and [5-8] matches any of the digits 5, 6, 7, or 8. If the dash sign is placed at the beginning or the end of the character class, it loses its special meaning; hence [+-] matches the plus or minus sign. The metacharacters `*`, `+`, `?`, `$`, `(`, and `)` lose their special meanings within character classes.

Complementary Character Classes ([^...]). If the first character in a character class is the caret (^), the character class is negated. Only those characters *not* in the character class will produce a match.

Logical OR (|). Two or more *expressions* separated by | will cause matches for any of the component expressions. Hence the regular expression `abc|lmn|xyz` will match either '`abc`,' '`lmn`,' or '`xyz`.'

Parenthesis Grouping ( ). Expressions may be logically grouped using parentheses. Hence, the expression `(abc)+` means one or more occurrences of the string '`abc`'. Multiple levels of grouping are possible such as `((abc|DEF)+(xyz))+` — which matches strings such as "`abcabcxyz`" and "`abcxyzDEFDEFabcxyz`."

EXAMPLES

A	match letter 'A'
^A	match letter 'A' at the beginning of a string
A$	match letter 'A' at the end of a string
.	match any character (including space or tab)
A+	match one or more instances of letter 'A'
A?	match a single instance of 'A' or the null string
A*	match one or more instances of 'A' or the null string
.*	match any string, including the null string
A.*B	match any string starting with 'A' up to and including 'B'
A|B	match 'A' or 'B'
(A)|(B)	match 'A' or 'B'
[AB]	match 'A' or 'B'
[^AB]	match any character other than 'A' or 'B'
AB	match 'A' followed by 'B'
AB+	match 'A' followed by one or more 'B's
(AB)+	match one or more instances of 'AB', e.g. ABAB
(AB)|(BA)	match 'AB' or 'BA'
A\{5\}	match five instances of 'A'
A\{5,\}	match five or more instances of 'A'
A\{5,9\}	match between five and nine instances of 'A'
[^A]AA[^A]	match two 'A's preceded and followed by characters other than 'A's
^[^^]	match any character at the beginning of a record except the caret
A +	match one or more lines containing the letter 'A'
A ?	match zero or one line containing the letter 'A'
AA+ *	match zero or more lines containing at least two consecutive 'A's

Examples of regular expressions.

VARIANTS

Three variants of regular expression syntax exist: (1) basic, (2) extend, and (3) Humdrum-extended. "Basic" regular expressions include the following features: the single-character wild-card (.), the repetition operator (*) but not (?) or (+), the context anchors (^) and ($), character classes ([...]), and complementary character classes ([^...]). Parenthesis-grouping is supported in Basic regular expressions, but the parentheses must be used in conjunction with the backslash to *enable* this function (i.e. \ (\ )).

"Extended" regular expressions include the following (added features are highlighted in bold): the single-character wild-card (.), the repetition operators (*), (?) and (+), **min-max character repetition** (\{ \}), the context anchors (^) and ($), character classes ([...]), complementary character classes ([^...]), **character-class keywords ([[:...:]]), the logical OR** (|), and parenthesis-grouping.

The Humdrum-extended syntax allows the defining of patterns spanning more than one line or record. When the metacharacters +, *, or ? appear at the end of a record, preceded by a tab character, they are **record-repetition operators** and pertain to the number of records, rather than the number of repetitions of the expression within a record. For example, the letter '`A`' followed by a tab, followed by + means one or more records containing the letter '`A`'. (This syntax is only available with the Humdrum **pattern** command.)

COMMANDS

A number of Humdrum commands make use of regular expression syntax, including **correl**, **fields**, **fill**, **hint**, **mint**, **num**, **patt**, **pattern**, **recode**, **regexp**, **rend**, **scramble**, **xdelta**, **yank**, and **ydelta**.

In addition, the following UNIX commands make use of regular expression syntax:

awk	pattern-action language
ed	line-oriented text editor
ex	text editor
expr	shell expression evaluator
grep	pattern-match command
gres	pattern substitution command
pg	interactive text display
sed	stream editor
vi	interactive full-screen text editor

UNIX commands employing regular expressions.

SEE ALSO

awk (UNIX), **ed** (UNIX), **egrep** (UNIX), **ex** (UNIX), **expr** (UNIX), **extract** (4), **fields** (4), **grep** (UNIX), **gres** (UNIX), **humsed** (4), **patt** (4), **pattern** (4), **pg** (UNIX), **recode** (4), **rend** (4), **sed** (UNIX), **vi** (UNIX), **xdelta** (4), **yank** (4), **ydelta** (4)

Section 6

Humdrum Special Files

Documentation Style

This section of the *Reference Manual* is intended to describe various special-purpose files associated with the Humdrum Toolkit. This section of the documentation will be distributed with a future release of Humdrum.

Section 7

Development Reference

Command Documentation Style

This section of the *Reference Manual* offers advice and direction for those users wanting to expand or tailor Humdrum so as to better suit a given application. Two types of extensions are possible. First, the user may define one or more new representation schemes that better represent the types of information of interest. For example, the user might define a new Humdrum representation scheme suitable for representing North Indian *tabla bols*. Second, users might wish to develop new software tools that manipulate one or more Humdrum representations in some fashion. For example, the user might create a command that identifies the roots of chords.

The ensuing section is divided into two parts. The first part *(Representation Development)* provides guidelines for defining new Humdrum representations. The second part *(Software Development)* provides tips for writing adjunct software; a standard *program skeleton* is described.

1. Humdrum Representation Development

1.1 Representation Assumptions

Three principle assumptions underly the Humdrum syntax. As with all design assumptions, these principles inevitably act as limitations — circumscribing what is possible.

The first assumption is that the information can be adequately represented using discrete rather than continuous signifiers. This is a limitation of all *symbolic* as opposed to *analogic* representations. If continuous data (such as conducting gestures, or analog sound) are to be represented using Humdrum, the data must be somehow transformed into a discrete form.

The second assumption is that information can be meaningfully organized as ordered successions of data tokens. More concretely, it is assumed that data can be arranged in linear *spines*, and that these spines are interpretable. This limitation implies that data tokens can be meaningfully ordered. Normally, *time* (as in sequences of events), or *space* (as in successive printed signs) provide suitable ordering devices. However, events that are conceived as entirely independent of one another are difficult to represent in Humdrum.

The third assumption is that the ASCII character set provides sufficient richness to act as an appropriate set of signifiers. In practice, this third assumption proves to be the most limiting. The use of alphabetic and numeric characters is itself of little concern; as Saussure pointed out, the choice of symbols is arbitary and conventional. Of greater concern is the limitation of size. There are some 128 characters in the basic ASCII set — not all of them are printable.

The size restriction imposed by the ASCII character set might be circumvented by using binary, pictoral or other signifiers. However, there are currently significant advantages to using ASCII signifiers. The principal advantage is that many important software tools already exist for the manipulation of ASCII text. The principal disadvantage of ASCII is addressed in Humdrum by allowing users to recycle the ASCII characters an indefinite number of times — through the use of new exclusive interpretations. Note that if all representation schemes use ASCII signifiers, then any software tool developed for the manipulation of ASCII text can be applied immediately to any Humdrum representation. This approach discourages the explosive proliferation of specialized software (each dealing with a unique representation), and encourages users to rely on a smaller toolkit consisting of more generalized and flexible tools. More precisely, this approach reduces the demands for software development while maximizing the range of tasks to which a user's skills may be applied.

An alternative to Humdrum's provision for multiple representations would be to provide a single representation, with greater contextual constraints on the positioning of signifiers. Unfortunately, high levels of context dependency end up placing an inordinate burden on software development and use. Humdrum attempts to simplify software development by minimizing as much as possible context-dependent meanings for signifiers.

1.2 Creating New Humdrum Representations

The Humdrum syntax provides a framework within which different music-related symbol-systems can be defined. Each symbol system or representation *scheme* is denoted by a unique *exclusive* *interpretation.* There is no restriction on the number of schemes that can be used or created within Humdrum. Humdrum users are consequently free to design new representation schemes that address various needs.

Many newly created representation schemes are apt to be *user-specific* — ways of organizing or representing information that are not likely to be of much value to other Humdrum users. In other cases, a well crafted representation scheme will prove to be of benefit to a wider community of users. Whether a new Humdrum representation scheme is intended for private use or for public distribution, it is prudent to develope good design habits. There is no such thing as a "perfect" representation, but it is possible to distinguish poor representations from better representations.

There are a number of considerations involved in the lucid design of a Humdrum interpretation. In general, the procedures involved in representation design can be summarized as follows:

1. Identify as clearly as possible the goal or goals you hope to achieve using this new representation. Formulate a list of questions you expect to be able to answer or address.
2. Create a list of the *essential signifieds* (concepts to be represented) that are needed to pursue the goal(s).
3. Make a supplementary list of *related signifieds* that seem peripheral to the immediate goal(s), but might prove important in pursuing related goals.
4. By examining the lists of essential and related signifieds, try to identify and define the *class* or *classes* of information with which you are dealing.
5. Identify whether the signifieds of immediate interest belong together in a single representation, or whether they ought to be split into two or more Humdrum interpretations.
6. Having established a clearer understanding of the class of signifieds, trim the previous lists of signifieds to a single list.
7. Identify those signifieds that may take many variant forms (as, for example, musical ornament symbols). Determine whether these variant forms are *finite, infinite* or *unknown* in number. If the number is infinite or unknown, a single signifier should be used — with provision for an auxilliary representation that can be used to identify the specific variant form.
8. Assign signifiers to all signifieds using the supply of available ASCII characters. In general, assign only a single character for each signified. Using individual ASCII characters as signifiers helps eliminates context dependency, and so reduces the complexity of the software needed to process the representation.
9. In assigning signifiers to signifieds, try to avoid English language initialisms (such as Q for quarter-note), since these mappings are poor mnemonics for non-English-speaking users.

Some of the "do's" and "dont's" of designing a Humdrum interpretation can be illustrated by considering a hypothetical representation problem. Below, we consider how to design a representation for keyboard fingering.

1.3 Defining a New Humdrum Interpretation: A Sample Problem

Suppose that we are interested in providing a comprehensive representation scheme for fingering keyboard instruments. More concretely, let's suppose that we are interested in studying the degree of performance difficulty for various keyboard works. Our *goal* might be to compare the relative difficulty of works by different composers, or whether a given musical arrangement is easier to perform than another arrangement. We might imagine, for example, writing a program which, given some knowledge of performance constraints, could accept fingering information as input and produce as output an index of the degree of difficulty. At a later stage, we might imagine creating an "intelligent" fingering program that could be used to determine an optimum way of fingering a particular keyboard passage. (Such a program might even be designed to take into account the unique physiological abilities, constraints, or preferences of a given performer.) In summary, our first research goal would use our "fingering" representation as an input to some analysis program, whereas our second research goal might produce the "fingering" representation as output.

Having identified the above goals, our next task is to identify the *essential signifieds* that would be needed to solve these problems. The most basic information we want to represent includes:

1. The identity of the finger and hand used in each key-press.
2. The identify of the key used in each key-press.
3. The order or sequence of key-presses.
4. The timing of each activity or movement.

Having identified what we see as the essential signifieds, we ought to pause and consider related signifieds that, although they appear to be peripheral to our goals, might prove important in pursuing related goals. By thinking ahead about these other signifieds, we might avoid future difficulties should we discover that another item of information proves crucial to our enterprise.

Some potential properties or attributes that we might consider representing could include the following:

1. The force or velocity with which the key is pressed.
2. Whether the hands are crossed — and if so, which arm is placed above the other.
3. What part of the finger/hand is used to press the key (e.g. knuckles).
4. Whether more than one finger is used to press the same key together.
5. Whether one (or more) finger is substituted for another finger in the course of holding a depressed key.
6. Whether trills are notated as a precise sequence of finger presses, or whether they are recorded as generic "trills."
7. Whether "Bebung" is used — that is, whether lateral or vertical pressure is applied once the key is depressed.
8. Whether a second performer can be accommodated (as in the case of a piano duet).

9. Physiological or anatomical attributes of the performer (such as hand-spans).
10. Pedalling.

A list of related signifieds is rarely likely to be exhaustive or complete. So it is important to take time to consider other possible types of information that relate to keyboard fingering.

In the initial stages of designing a Humdrum representation, it is generally wise to try to formulate a fairly exhaustive list of possible pertinent attributes. The purpose of such a list is to ensure that an informed decision is made regarding those properties we wish to include in the representation, and those properties we propose to exclude. More specifically, our goal is to exclude information on the basis of an explicit decision rather than due to a tacit oversight.

Given the above list of potential signifieds, the next step is to pause and consider the nature of the *class* of information we wish to deal with. For example, the idea of "pedalling" raises an interesting representation question. Are we trying to represent keyboard *fingering?* Or is our task the representation of keyboard *performance*? Pedalling is obviously part of keyboard performance, but not something fingers do. Are we mistaken in thinking that our representation task is limited to fingering? Also, since larger arm and body movements are essential aspects of good performance, should we also consider representing these additional factors?

In light of our goal of measuring performance difficulty, we would have to admit that pedalling can indeed contribute to the physical challenge arising from performing a given work. In terms of our research task therefore, it makes sense to include pedalling information. However, we might balk at the prospect of mixing fingering and pedalling within a single representation — especially since some keyboard instruments (e.g. clavichord) have no pedals. Moreover, the pedals on a piano differ considerably from the pedals on an organ, although both contribute to overall performance difficulty.

At this point, we are invited to consider whether the various signifieds in the above list truly belong together in a single *representation,* or whether they ought to be split into two or more Humdrum interpretations. The above discussion suggests that we might distinguish at least five classes of information: *performance (broadly construed), fingering, body movement, pedal-boarding* (as on the organ), and *pedalling* (as on the piano or harpsichord). Moreover, we might define "performance information" as the combination of fingering plus pedalling or pedal-boarding. In short, it would make sense to define *three* representations: fingering, pedalling, and pedal-boarding, and to assume that our program measuring performance difficulty will accept any combination of one or more of these three classes of information.

Our task has clearly expanded somewhat, since now we need to consider three types of representation rather than one. For the purposes of this tutorial example, we might set aside the problems of representing pedalling and pedal-boarding and focus on the fingering aspect of keyboard performance.

Now that we have a clearer understanding of the class of signifieds, we can begin to trim the lists of signifieds to a single short list. We have decided not to represent pedalling using the same Humdrum interpretation as for fingering. We might also decide that the fingering activity for a second performer can be represented using a second independent spine of information. We might also dispense with representing the force or velocity of key-depression, and what part of the finger/hand is used to press the key. We might have decided to represent the fingering for trills, but not to encode each key-stroke of the trill separately. We could also decide that representing physiological attributes of the performer (such as hand-spans) ought to be left as a separate representation. Finally, we might have also decided not to represent *Bebung*. This leaves a trimmed list of eight types of signifieds:--

1. The identity of the finger and hand used in each key-press.
2. The identify of the key used in each key-press.
3. The order or sequence of key-presses.
4. The duration of each activity or movement.
5. Whether the hands are crossed — and if so, which arm is placed above the other.
6. Whether more than one finger is used to press the same key together.
7. Whether finger substitution occurs.
8. The fingering for trills.

Before going on to map signifiers and signifieds we need to consider those signifieds that may take variant forms. What sort of "variations" might appear in a fingering representation? An obvious form of variation occurs when alternative fingerings are possible — that is, where a passage contains two (or more) ways of assigning key-presses to different fingers. Our first task here is to determine whether the number of variant forms is finite, infinite, or unknown in number. We can consider this question both at the level of the individual key-press, and at the level of the entire work. In the case of the individual key-press, the maximum number of variants is ten — since there are no more than ten fingers. If more than one finger is used to press a key, or if finger-substitutions occur, then the maximum number of variants is somewhat more than ten — although still finite in number. At the level of the entire work, the maximum number of variants is potentially very large (at least 10 × the total number of key-presses in the work). Nevertheless, this number remains a finite value for works of finite length. The question arises, do we want our fingering representation to represent a single performance of a keyboard passage, or is the representation intended to represent alternative forms of performance?

In order to answer this question we must return once again to our initial goals. In analysing the degree of performance difficulty for a work, we might prefer to analyse a single (actual or plausible) performance. Measuring the degree of performance difficulty for a *class* of variant performances is apt to prove difficult. On the other hand, the musical score for a keyboard work may contain no fingering indications whatsoever. Therefore it would be wrong to assume that a single fingering specification would give an accurate indication of the performance difficulty for a given musical work. This raises the question of whether our intention is to measure the performance difficulty of a specific sequence of key-presses used in a performance, or whether our intention is to measure the performance difficulty of a particular musical work.

As noted above, measuring the difficulty for a class of variant performances is likely to prove difficult. There are many many ways of fingering a keyboard work. Averaging the performance difficulty for the complete class of possible fingering arrangements would appear silly since most of these fingerings would be awkward. One could argue, for example, that the degree of performance difficulty for a work can be best established by analysing the single most convenient way of fingering the work. Alternatively, one could argue that the degree of performance difficulty for a work can be determined by examining a handful of the most convenient ways of fingering the work. Out of the large number of *possible* fingerings, it is only the *plausible* fingerings that really count.

Whether our intention is to measure a single fingering sequence or a class of such fingerings, a helpful question to consider here ~~is~~ where do we expect to get our fingering data? Three sources come to mind: (1) recorded fingering data from an actual performance, (2) fingerings (including alternatives) notated in printed scores or annotated by keyboard performers, and (3) fingering data (including alternatives) generated by a computer program. In the case of an actual performance, there will be only one fingering. The other sources can potentially produce more than one fingering at a time.

Having identified those attributes that we wish to represent, we need to consider how the representation ought to be structured. Specifically, we need to consider how our fingering representation can be coordinated with other Humdrum interpretations. The most important coordination task is ensuring that our representation will correspond well with the core `**kern` representation. Each data record in `**kern` represents a single sonority — a moment in time that differs from the previous state. Since key-presses are closely related to notes, we might want to coordinate each of the `**kern` note tokens with possible key-presses. Many Humdrum pitch-related representations include barlines — which are useful markers for coordinating such representations. This suggests that it might be useful to include barlines in our fingering representation. Given both the barline and note-token/key-press correspondences, we should have little difficulty ensuring that our fingering representation will be fully coordinated with a number of other Humdrum representations.

We are now ready to consider how to map our signifieds with a set of appropriate signifiers. In general, we should endeavor to define one signifier for each attribute. First, consider how we might identify the individual fingers. A good system would be to identify each finger by a unique signifier — such as a unique decimal integer. However, there is a long-standing tradition of identifying the thumb of each hand as the number "1", the index finger by the number "2" and so on. Given the limited number of fingers, some context-dependency may be appropriate here. In short, we may decide to identify specific fingers through a *ligature* of "hand+finger" — e.g. left-3 or right-5. The signifiers "left" and "right" obviously introduce an English bias. It would be better to consider more universally recognized terms such as "mano destra" (MD) and "mano sinistra" (MS). However, not every user will find these is terms familiar or comfortable.

We need not rely on a literalism or initialism. "Left" and "right" are concepts that lend themselves well to pictorial representation, so we might consider using those ASCII characters that convey a left-right pictorial dimension. Possible contenders would include various letter-contrasts: `d` versus `b`, `J` versus `L`; the three types of parentheses: `(` versus `)`, `{` versus `}`, and `[` versus `]`; and the greater-than and less-than signs: < versus >. The letter contrasts `J` versus `L` are especially poor since although the angle of the letter `L` is drawn to the right, "L" implies an initialism for "left" — and so is apt to cause confusion. The parentheses cannot be misconstrued as literalisms or initialisms, so they are somewhat better signifiers. However the greater-than and less-than signs are the mostly clearly arrow-like, and so perhaps provide a better pair of left-right signifiers.

Having decided upon the signifiers for left and right, and having adopted the tradition of numbering the fingers (1=thumb, 2=index, 3=middle, 4=ring, 5=little), we could continue mapping signifiers to signifieds, taking care to minimize context dependency.

In reflecting on the above discussion, readers are apt to feel that one or another type of information ought to have been included, or that the signifiers ought to be assigned in a different manner. Since Humdrum provides a framework within which alternative representation schemes can be designed, there is no need to defend a given representation from competing schemes. Whether an interpretation survives and proliferates will be determined, not by its conceptual elegance or completeness, but by whether it is found to have a practical utility in solving users' problems.

2. Humdrum Software Development

The Humdrum toolkit is necessarily limited in scope and there are many functions that users will wish to add. In developing adjunct software tools, it is imperative that the software conform to the following design conventions:

1. Programs should be general-purpose and adapt to a wide variety of input circumstances.
2. If possible, programs should be able to process any Humdrum input rather than be limited to a given type of input interpretation.
3. Command names should be limited to 8 characters in length in order to ensure portability to DOS systems.
4. Command names should preferably be the same as the output interpretation produced by the command.
5. Command names should not be unduly abbreviated since infrequently used software is less easily remembered than frequently used system commands.
6. The command syntax should conform to standard POSIX conventions.
7. Errors and warnings should be prefaced by giving the name of the program or command which issues the error message. e.g.

   ```
   vox: ERROR: voice 3 begins with a null token.
   ```

8. Errors messages should be sent to "stderr" rather than to the standard output.
9. Wherever possible, 'filter' programs should produce outputs that are identical in structure to the input. More specifically, input line numbers should correspond to output line numbers — where appropriate.
10. Comments, interpretations, barlines, and double barlines should be echoed in the output as the default condition (except in the case of formatted non-Humdrum outputs).
11. For many programs, the user should be able to skip the processing of certain types of tokens (such as barlines) by specifying a **-s** flag — followed by a user-defined regular expression. Tokens matching the regular expression should be echoed unprocessed in the output stream.
12. Programs should handle spine-path changes in a fashion appropriate to the nature of the command.
13. Comments and interpretations should be identified by explicitly matching the exclamation mark or asterisk in the *first* column of the input data token. Exclamation marks and asterisks are legitimate data signifiers when not occurring in the first column of an input token.
14. Where possible, outputs should *not* be formatted with descriptive labels etc. The preferred output format is to have all outputs conform to the Humdrum syntax. This ensures that all outputs can themselves be used as inputs to other Humdrum programs.
15. Programs should generally avoid assumptions concerning context-dependent inputs. Inputs should be assumed to be context-free.

16. Programs should be able to handle inputs with unexpected user extensions or representational addenda — such as the presence of spurious or unknown characters.

17. Programs that search or examine inputs for certain features, properties, or errors should return a *null* output if nothing is found. Messages indicating that 'nothing was found' should be avoided. "Silence is golden."

2.1 Standard Program Skeleton

Much of the Humdrum software was originally developed using the AWK programming language. AWK was designed by Alfred Aho, Brian Kernighan, and Peter Weinberger.† It is syntactically very similar to the C programming language, but is easier to use and promotes better software productivity. AWK provides powerful text manipulation features that make it admirably suited to the creation of Humdrum software. AWK is also a very easy language to learn, and is an excellent first language for novice programmers.

The Humdrum Toolkit includes programing skeletons that may provide a useful starting place for software development using AWK. Two skeleton files are provided with the toolkit: `skeleton.ksh` and `skeleton.awk`. The kornshell file (`.ksh`) parses the command line, issues appropriate error messages if the command is improperly invoked, displays a help screen if necessary, and assembles the command parameters to invoke an awk script for the command (`.awk`).

The `skeleton.awk` skeleton contains a main loop that is normally executed for each record of input. A series of useful functions are included in the AWK skeleton program. These functions include:

Parse_command. This function checks that the input passed from the corresponding kornshell script for the command. The Parse_command function contains a list of valid options and assigns the passed parameters to the appropriate option variables.

Store_indicators. This function allows the spine-path indicators for the current record to be stored in the array `path_indicator` so that they may be used later.

Store_new_interps. This function stores the new interpretations found in an interpretation record for each spine.

Process_indicators. This function takes the spine-path indicators that were stored in the array 'path_indicator' in the function 'store_indicators' and manipulates the arrays 'path_indicator' and 'current_interp' according to the contents of the array 'path_indicator'.

Ins_array_pos. This function inserts new positions in the arrays 'path_indicator', 'current_interp', and 'current_key' and copies elements so that everything is preserved

Del_array_pos. Performs the opposite of function 'ins_array_pos'.

Exchange_spines. This function exhanges two spines by exchanging the corresponding elements in current_interp.

† Aho, A., Kernighan, B. & Weinberger, P. *The AWK Programming Langauge.* Reading, Massachusetts: Addison-Wesley Publishing Co. 1988.

Bibliography

Aho, A., Kernighan, B. & Weinberger, P.
The AWK Programming Langauge.
Reading, Massachusetts: Addison-Wesley Publishing Co. 1988.

Bregman, A. S.
Auditory Scene Analysis; The Perceptual Organization of Sound.
Cambridge, Mass.: M.I.T. Press, 1990.

Brinkman, A.
Pascal Programming for Music Research.
Chicago: University of Chicago Press, 1990.

Butler, D.
"Describing the perception of tonality in music: A critique of the tonal hierarchy theory and a proposal for a theory of intervallic rivalry," *Music Perception,* Vol. 6, No. 3 (1989) pp. 219-242.

Christ, W., Delone, R., Kliewer, V., Rowell, L. & Thomson, W.
Materials and Structure of Music.
Englewood Cliffs, N.J.: Prentice-Hall, 1966; third edition, 1980; in two volumes.

Cole, H.
Sounds and Signs; Aspects of Musical Notation. London: Oxford University Press, 1974.

Cope, D.
New Music Notation. Dubuque, Iowa: Kendall/Hunt Publishing Co., 1976.

Erickson, R.
"The DARMS project: A status report." *Computers and the Humanities,* Vol. 9 ((1975) pp. 91-298.

Dannenberg, R.
"Music representation issues, techniques, and systems." *Computer Music Journal,* Vol. 17, No. 3 (1993) pp. 20-30.

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

Greenwood, D. D.
"Critical bandwidth and the frequency coordinates of the basilar membrane," *Journal of the Acoustical Society of America,* Vol. 33, No. 4 (1961) pp. 1344-1356.

Hall P. & Dowling, G.
"Approximate string matching," *ACM Computing Surveys,* Vol. 12 (1980) pp. 381-402.

Hewlett, W. B. & Selfridge-Field, E. (editors)
Directory of Computer Assisted Research in Musicology 1988.
Menlo Park: Center for Computer Assisted Research in the Humanities, 1988.

Hewlett, W. B. & Selfridge-Field, E. (editors)
Computing in Musicology; A Directory of Research;
Menlo Park: Center for Computer Assisted Research in the Humanities, 1989.

Hewlett, W. B. & Selfridge-Field, E. (editors)
Computing in Musicology; A Directory of Research;
Volume 6. Menlo Park: Center for Computer Assisted Research in the Humanities, 1990.

Hewlett, W. B. & Selfridge-Field, E. (editors)
Computing in Musicology; A Directory of Research;
Volume 7. Menlo Park: Center for Computer Assisted Research in the Humanities, 1991.

Hewlett, W. B. & Selfridge-Field, E. (editors)
Computing in Musicology; An International Directory of Applications;
Volume 8. Menlo Park: Center for Computer Assisted Research in the Humanities, 1992.

Horwood, F. J.
The Basis of Harmony.
Toronto: Gordon V. Thompson Ltd, 1948.

Huron, D.
"Error categories, detection, and reduction in a musical database," *Computers and the Humanities,* Vol. 22, No. 2 (1988) pp. 253-264.

Huron, D.
"Voice denumerability in polyphonic music of homogeneous timbres," *Music Perception,* Vol. 6, No. 4 (1989) pp. 361-382.

Huron, D.
"Characterizing musical textures," *Proceedings of the 1989 International Computer Music Conference,* San Francisco: Computer Music Association, 1989; pp. 131-134.

Huron, D.
"The avoidance of part-crossing in polyphonic music: Perceptual evidence and musical practice," *Music Perception,* Vol. 9, No. 1 (1991) pp. 93-104.

Huron, D.
"Harmony: A Psychoacoustical Approach [review of]," *Psychology of Music,* Vol. 19, No. 2 (1991) pp. 173-177.

Huron, D.
"Tonal consonance versus tonal fusion in polyphonic sonorities," *Music Perception,* Vol. 9, No. 2 (1991) pp. 135-154.

Huron, D.
"Design principles in computer-based music representation," in: A. Marsden & A. Pople (editors), *Computer Representations and Models in Music.* London: Academic Press, 1992; pp. 5-59.

Huron, D. & Fantini, D.
"The avoidance of inner-voice entries: Perceptual evidence and musical practice," *Music Perception,* Vol. 7, No. 1 (1989) pp. 43-47.

Huron, D. & Parncutt, R.
"An improved model of tonality incorporating pitch salience and echoic memory," *Psychomusicology*, in press.

Johnson-Laird, P.
"Rhythm and meter: A theory at the computational level," *Psychomusicology*, Vol. 10 (1991) pp. 88-106.

Kameoka, A. & Kuriyagawa, M.
"Consonance theory, part I: Consonance of dyads," *Journal of the Acoustical Society of America*, Vol. 45, No. 6 (1969a) pp. 1451-1459.

Kameoka, A. & Kuriyagawa, M.
"Consonance theory, part II: Consonance of complex tones and its calculation method," *Journal of the Acoustical Society of America*, Vol. 45, No. 6 (1969b) pp. 1460-1469.

Kochan, S. & Wood, P.
UNIX Shell Programming.
Hasbrouck Heights, New Jersey: Hayden Book Co. 1985.

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

Krumhansl, C. & 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.

Kurkela, K.
Note and Tone; A semantic analysis of conventional music notation. Helsinki: Suomen Musiikkitieteellinen Seura (Musicological Society of Finland), 1986.

Longuet-Higgins, H. C., & Lee, C. S.
"The perception of musical rhythms," *Perception*, Vol. 11 (1982) pp. 115-128.

Miller, G. A. & Heise, G. A.
"The trill threshold," *Journal of the Acoustical Society of America*, Vol. 22, No. 5 (1950) pp. 637-638.

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

Moore, B. & Glasberg, B.
"Suggested formulae for calculating auditory-filter bandwidths and excitations patterns," *Journal of the Acoustical Society of America*, Vol. 74, No. 3 (1983) pp. 750-753.

Mortice Kern Systems Inc.
The MKS Toolkit.
Mortice Kern Systems Inc., 35 King Street North, Waterloo, Ontario, Canada N2J 2W9.

Nettheim, N.
"On the accuracy of musical data, with examples from Gregorian Chant and German folksong." *Computers and the Humanities*, Vol. 27, (1993) pp. 111-120.

Newcomb, S.
"Standard music description language complies with hypermedia standard." *IEEE Computer,* Vol. 24, No. 7 (1991) pp. 76-79.

Newcomb, S., Kipp, N. & Newcomb, V.
"The HyTime hypermedia time-based document structuring language." *Communications of the ACM,* Vol. 34, No. 11 (1991) pp. 67-83.

van Noorden, L.
Temporal Coherence in the Perception of Tone Sequences.
Eindhoven: Druk vam Voorschoten, 1975.

Opolko, F., & Wapnick, J.
McGill University Master Samples. Montreal: McGill University, 1987, 1989.

Orpen, K. & Huron, D.
"Measurement of similarity in music: A quantitative approach for non-parametric representations," *Computers in Music Research,* Vol. 4 (1992) pp. 1-44.

Ottman, R. W.
Elementary Harmony; Theory and Practice.
Englewood Cliffs, N.J.: Prentice Hall, 1961; fourth edition, 1989.

Overill, R. E.
"On the combinatorial complexity of fuzzy pattern matching in music analysis," *Computers and the Humanities,* Vol. 27 (1993) pp. 105-110.

Parncutt, R.
Harmony: A Psychoacoustical Approach.
Berlin: Spinger-Verlag, 1989.

Parncutt, R.
"Review of Terhardt's psychoacoustical model of the root(s) of a musical chord," *Music Perception,* Vol. 6, No. 1 (1988) pp. 65-94.

Plomp, R. & Levelt, W. J. M.
"Tonal consonance and critical bandwidth," *Journal of the Acoustical Society of America,* Vol. 37, (1965) pp. 548-560.

Prather, R. & Elliot, R.
"SML: A structured musical language," *Computers and the Humanities,* Vol. 22 (1988) pp. 137-151.

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

Rastall, R.
The Notation of Western Music. London: J. M. Dent & Sons Ltd., 1983.

Riemenschneider, A.
371 Harmonized Chorales and 69 Chorale Melodies with Figured Bass by Johann Sebastian Bach.
New York: G. Schirmer, 1941.

Sandell, G.
Concurrent timbres in orchestration: A perceptual study of factors determing "blend." PhD dissertation, Northwestern University, Evanston, Illinois, 1991.

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.

Stone, K.
Music Notation in the Twentieth Century; A Practical Guidebook. New York: W. W. Norton & Co., 1980.

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

Terhardt, E.
"Calculating virtual pitch," *Hearing Research,* Vol. 1 (1979) pp. 155-182.

Terhardt, E., Stoll, G. & Seewann, M.
"Algorithm for extraction of pitch and pitch salience from complex tonal signals," *Journal of the Acoustical Society of America,* Vol. 71 (1982) pp. 679-688.

Terhardt, E., Stoll, G. & Seewann, M.
"Pitch of complex signals according to virtual-pitch theory: test, examples, and predictions," *Journal of the Acoustical Society of America,* Vol. 71 (1982) pp. 671-678.

Thomas, R. & Yates, J.
A User Guide to the UNIX System.
Berkeley, California: Osborne/McGraw-Hill 1982.

Thomassen, J.
"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.

Vos, J. & van Vianen, B.
"The effect of fundamental frequency on the discriminability between pure and tempered fifths and major thirds," *Perception & Psychophysics,* Vol. 37 (1985) p.509.

Wiggens, G., Miranda, E., Smaill, A. & Harris, M.
"A framework for the evaluation of music representation systems," *Computer Music Journal,* Vol. 17, No. 3 (1993) pp. 31-42.

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

Zwicker, E., Flottorp, G. & Stevens, S. S.
"Critical bandwidth in loudness summation," *Journal of the Acoustical Society of America,* Vol. 29, No. 5 (1957) pp. 548-557.

INDEX

Entries in this index conform to the following typeface conventions: Humdrum commands and sub-command-level instructions are indicated in **boldface** print; Humdrum representations, system-level commands (such as UNIX commands), filenames, and variable names are all indicated in `typewriter` print; titles of musical works are printed in *italics*.

<space> suspend playback (**perform**) 377
$ in command line 230
$ last data record (**extract**) 280
$ last field anchor (**extract**) 281
$ last segment in input (**yank -r** *range*) 476
$-1 second-last data record (**extract**) 280
`$HUMDRUM/regexp.txt` test file 398
& substitute with matched string (**humsed**) 312
| regular expressions, logical OR 311
| UNIX pipe 228
+ next measure (**perform**) 378
+ regular expressions, one-or-more 311
, reduce tempo (**perform**) 378
- repeat measure (**perform**) 378
-h Humdrum help screens 230
/ forward string search (**perform**) 377
< UNIX here-is file 229
< reduce tempo (**perform**) 378
> UNIX file redirection 229
> increase tempo (**perform**) 378
>> UNIX file append 229
= next measure (**perform**) 378
? backward string search (**perform**) 377
? in command line 230
? regular expressions, zero-or-more 311
_ repeat measure (**perform**) 378
***** in command line 230
`*+` tandem interpretation 212
`*-` tandem interpretation 212
`*>` tandem interpretation 188-189,210-211
`*^` tandem interpretation 212
`*|.` tandem interpretation 214
`*[a-gA-G][-+]*` tandem interpretation — key signature 105,461
`*a2` tandem interpretation 187
`*all` tandem interpretation 184
`*AT:` tandem interpretation 76,190
`*Ch` tandem interpretation 208
`*clef` tandem interpretation 105,185-186
`*conct` tandem interpretation 187
`*ennun` tandem interpretation 162
`*ensemb` tandem interpretation 187
`*FT:` tandem interpretation 78,190
`*H` tandem interpretation 192
`*I` tandem interpretation 193-198
`*I` tandem interpretation — instrument 95,105
`*IC` tandem interpretation 199
`*IC` tandem interpretation — instrument class 95,105
`*ICbras` tandem interpretation 199
`*ICidio` tandem interpretation 199
`*ICklav` tandem interpretation 199
`*ICstr` tandem interpretation 199
`*ICvox` tandem interpretation 199
`*ICww` tandem interpretation 199
`*ITr` tandem interpretation 105,221,460-461
`*k[ ]` tandem interpretation — key 105,461
`*k` tandem interpretation 202-203
`*K` tandem interpretation 202-203

*L tandem interpretation 204-205
*M tandem interpretation — meter signature 95,105
*MM tandem interpretation 206-207,217-218
*MM tandem interpretation — tempo 95,105
*ripien tandem interpretation 187
*RT: tandem interpretation 77,190
*S- tandem interpretation 216,434,437
*S/1 tandem interpretation 216
*S/*number* tandem interpretation 437
*S/fin tandem interpretation 216,434,437
*S/name tandem interpretation 216
*solo tandem interpretation 187
*staff tandem interpretation 213
*strophe tandem interpretation 215-216,434,437
*tb tandem interpretation 220
*tb tandem interpretation — time-base 105
*thru tandem interpretation 219
*thru, tandem interpretation 448
*Tr tandem interpretation 221,459-460
*ueber tandem interpretation 209
*unter tandem interpretation 209
*v tandem interpretation 212
*vibrato tandem interpretation 162
*x tandem interpretation 212
**accid, sample interpretation 242
**bowing 12
**Cdev, sample interpretation 242
**cents representation 46-48
**cents, translate to 237
**correl representation 49-50
**dB representation 56-57
**date representation 51-55
**date, in reference records 30-32,35
**deg representation 58-60
**deg, translate to 261-264
**degree representation 12,61-63
**degree, translate to 265-268
**diaton, sample interpretation 242
**diss representation 64-65
**dur representation 66-68
**dynam representation 12
**embel representation 69-72
**freq representation 12,73-75
**freq, translate to 294-296
**fret representation 76-84,190-191
**harm representation 12,85-90
**hint representation 184
**IPA representation 91-95
**IPA representation summary 92
**iv, translate to 373-375
**kern representation 15,96-106
**kern representation summary 99
**kern, general file information 235-236
**kern, translate to 323-327
**melac representation 107-109
**metpos representation 110-112
**MIDI 339-344,376-380,394-396,425-426,445
**MIDI representation 113-115
**MIDI, translate to 339-343
**mint representation 12,116-118
**nf, translate to 373-375
**oct, sample interpretation 242
**ordo representation 119-120
**pc representation 121-124
**pc, translate to 369-372
**pcset representation 12,125-129
**pcset, translate to 373-375
**pitch representation 12,130-133
**pitch, translate to 383-386
**recip representation 12,97,134-136,206
**semits representation 137-139
**semits, translate to 415-417
**simil representation 140-141
**simxrf representation 419
**solfa representation 142-144
**solfa, translate to 427-430
**solfg representation 145-148
**solfg, translate to 431-433
**specC representation 149-151
**spect representation 152-153
**synco representation 154-156
**takt representation 157-159
**text representation 12,160-164
**time representation 165-166
**Tonh representation 167-170
**Tonh, translate to 453-456
**vox# representation 174-175
**URrhythm representation 171-173
**Zeit representation 176-179
**Zeit, in reference records 29,31
12 tone row (**reihe**) 399-403
2-3 retardation (**embel) 70
330 MIDI card address 379,395,445
4-3 suspension (**embel) 70
7-6 suspension (**embel) 70
7-8 retardation (**embel) 70
8va bassa 185
9-8 suspension (**embel) 70

A

a append (**humsed**) 310
abbreviated data format 436,438
abbreviated data format (**thru**) 446-450
abbreviated format 188-189,219
absolute scale degree, representation of (**degree) 61-63
absolute time period, representation of (**Zeit) 176-179
accent, representation of melodic (**melac) 107-109
accented lower neighbor (**embel) 70
accented passing tone (**embel) 70
accented upper neighbor (**embel) 70
accents, language diacriticals 28
accents, pitch-related (**melac**) 332-333
accents, representation of (**kern) 97
acciaccaturas, representation of (**kern) 98

accidentals, incoherent (**proof**) 387
accidentals, missing precautionary (**proof**) 387
accidentals, representation of (`**kern`) 96
accordion 195,197
acoustic spectra 8
act number, reference records 31
added seventh (`**embel`) 70
address, MIDI card 379,395,445
Aeolian mode 458
Aho, A. 519-520
Albanian language 35
alias/stage name, reference records 29
Alkan, C. 189
alternative concurrent information (**strophe**) 434
alternative title, reference records 30
alto clef 185
alto voice 194
amalgamate articulation information (**cleave**) 243
amalgamate files 232-234
amalgamate pitch and duration (**cleave**) 240
American pronunciation (`**IPA`) 91-94
ametric meter indications 206
amplitude, representation of (`**dB`) 56-57
anacrusis passages, extraction of 478
analytic information 34,96-97
anchors, regular expressions 496,503
annularis finger (`**fret`) 79,81
Anon., names 27
anonymous, names 27
ANSI standard pitch notation 383
anticipation (`**embel`) 70
append (a), **humsed** 310
appoggiaturas (`**embel`) 70
appoggiaturas, representation of (`**kern`) 98
Arabic language 27,35
archlute 194
Armenian language 35
arpeggiation, representation of (`**kern`) 97
arpeggio tone (`**embel`) 70
arpeggio tones, grouping into chords 245
arranger, reference records 29
articulation marks, incoherent (**proof**) 387
articulation marks, representation of (`**kern`) 96-97
artificial harmonics (`**fret`) 78,81
ascending slur articulation (`**fret`) 80-81
ASCII 8,492
ASCII language setting, reference records 34
assemble (**encode**) 269
assemble full score 232-234
assemble command 232-234
associated works, reference records 34
asterisk in command line 230
asterisks 10
atonal tandem interpretation 200
attacca, representation of (`**kern`) 97
attributed composers 26
attributed composers, reference records 28
audition (**perform**) 376-380
augmentation dots, representation of (`**kern`) 98
authorship information 28-29
average tempo, calculation of (**midi**) 342
`awk` command (UNIX) 26,313,397-398,480,492
`AWK_VER` environment variable 398
Azeri language 35

B

B-A-C-H pattern 7
Bach, J.S. 13,59,62,103,107,111,158,165,256,350
backslash character in command line 230
backward string search (?), **perform** 377
bagpipes, Irish 195
bagpipes, Scottish 195
ballet steps 8
banjo 194
banjo, tablature representation 76
barbershop quartet, genre 34
barbershop quartets 7
baritone voice 193
barlines 15
barlines, avoiding repeated (**fill**) 293
barlines, common system 43-45
barlines, double 43
barlines, eliminate duplicate (**humsed**) 243
barlines, number of (`**kern`) 236
barlines, number of double (`**kern`) 236
barlines, representation of (`**kern`) 98,101
barlines, representation of double (`**kern`) 99
barlines, with measure numbers 44
barlines, with pauses 43
Baroque period (`**Zeit`) 178
Baroque, period 34
bars, extract specified (**yank**) 476-480
bars, numbering of (**num**) 354-358
bars, renumbering of (**num**) 354-358
Bartók, B., dissonances in 7
Bartók, Bela 335
basic regular expressions 501,505
bass clef 185
bass course 77
bass drum 197
bass guitar, tablature representation 76
bass voice 193
basso cantante voice 194
basso profondo voice 194
bassoon 195
beaming, representation of (`**kern`) 96-97
beat position, representation of (`**takt`) 157-159
bebop, style 34
Beethoven, L. van 174
bell 197
bémol 431
bending of pitch (`**fret`) 80-81
Bengali language 36,204
bibliographic information, files 26
binary files (**smf**) 425
binary form 188

biwa 194
blue staff line 214
Boehm, T. 178
Bondy, S. 2
bowing, col legno (**fret) 78,81
bowing, direction of (**fret) 79,81
bowing, ponticello (**fret) 78,81
bowing, representation of (**kern) 97
bowing, spiccato (**fret) 78,81
bowing, sul tasto (**fret) 78,81
bowing, tremolo (**fret) 78,81
Brahms, J. 143
Brandenburg Concertos 7
brass instruments 199
Bratsche 195
Bregman, A.S. 520
breve durations (**kern) 98
Brinkman, A. 520
British pronunciation (**IPA) 94
Bulgarian language 36
Butler, D. 520
BWV catalogue 32

C

C clef 185
caeruleus staff line 214
calliope 195
Cambodian language 36
canonical duration, representation of (**kern) 96
canonical duration, representation of (**recip) 134-136
canonical ordering of signifiers (**kern) 102-103
Cantonese language 36
capturing Humdrum data (**encode**) 269
cardinal range specification 476
carillon 197
castanets 197
castrato voice 194
cat command (UNIX) 478-479
catalogue, BWV 32
catalogue, Hoboken 32
catalogue, Köchel 32
CCARH 2
celesta 197
cembalo 194,197
census command 235-236
Center for Computer Assisted Research in the Humanities 2
centroid, representation of spectral (**specC) 149-151
cents, representation of (**cents) 46-48
cents, translate to 239
cents command 237-239
cents representation 46-48
ch'in 194
chalumeau 195
changing tones (**embel) 70
character classes, complementary 498,504
character classes, keywords in 499
character classes, metacharacters in 498,501
character classes, regular expressions 497,504
checksum validation number, reference records 33,465-466
chemical processes 8
Chief Falcon Feather 27
child's voice 193
chimes 197
Chinese, tonal inflections (**IPA) 91
chitarrone, tablature representation 76
Chopin, Frederick 28
chord alterations (**harm) 85
chord inversion (**harm) 85,87
chord root (**harm) 85,87
chord type (**harm) 85,87
chordal tones (**embel) 70
chordal tones, arpeggio 70
chords, French sixth (**harm) 87
chords, German sixth (**harm) 87
chords, Italian sixth (**harm) 87
chords, Neopolitan sixth (**harm) 87
chords, Tristan (**harm) 87
chords, augmented (**harm) 85,87
chords, augmented sixth (**harm) 87
chords, chromatic mediant (**harm) 85
chords, diminished (**harm) 85,87
chords, diminished seventh (**harm) 86-87
chords, eleventh (**harm) 86-87
chords, enharmonic spelling (**harm) 87
chords, ethnic sixth (**harm) 87
chords, extended tertian (**harm) 86-87
chords, first inversion (**harm) 86-87
chords, half diminished seventh (**harm) 86-87
chords, major (**harm) 85,87
chords, minor (**harm) 85,87
chords, ninth (**harm) 86-87
chords, root position (**harm) 86-87
chords, second inversion (**harm) 86-87
chords, secondary dominants (**harm) 86-87
chords, seventh (**harm) 86-87
chords, third inversion (**harm) 86-87
chords, thirteenth (**harm) 86-87
Christ, W. 520
chromatic lower neighbors (**embel) 70
chromatic mediant chords (**harm) 85
chromatic offset (**trans**) 457
chromatic transposition (**trans**) 457-461
chromatic unaccented passing tones (**embel) 70
chromatic upper neighbors (**embel) 70
ciphers (stuck notes) 445
citation, electronic 35
cittern 194
clarinet, piccolo, soprano, alto, bass 195
Classical period (**Zeit) 178
classify numerical data tokens (**recode**) 389-393
clavecin 194,197
clavichord 194,197
cleave command 240-244

clefs 185-186
Clift, S. 1
clock ticks (**midi**) 341
clock ticks, MIDI (**record**) 395
closed beams, representation of (`**kern`) 97
`cmp` command (UNIX) 102
`cnt` file extension 237
codas, through-composed expansions (**thru**) 446
Cole, H. 520
col legno bowing (`**fret`) 78,81
coloratura soprano voice 194
colored staff lines 214
command documentation 225
command line syntax 225
command line, example 226
command line, options 226
command line, problems with shell metacharacters 281
command line, use of boldface 225
command line, use of dash character 226
command line, use of ellipses 225
command line, use of italics 225
command line, use of square brackets 225
command, **assemble** 232-234
command, **census** 235-236
command, **cents** 237-239
command, **cleave** 240-244
command, **context** 245-253
command, **correl** 254-260
command, **deg** 261-264
command, **degree** 265-268
command, **diss** 1
command, **encode** 1,269-279
command, **extract** 280-285
command, **fields** 286-289
command, **fill** 290-293
command, **freq** 294-296
command, **hint** 297-302
command, **humdrum** 303-309
command, **humsed** 310-313
command, **humver** 314
command, **infot** 315-320
command, **iv** 321-322
command, **kern** 323-327
command, **key** 328-331
command, **melac** 332-333
command, **metpos** 334-338
command, **midi** 1,339-343
command, **midreset** 344
command, **mint** 345-351
command, **nf** 352-353
command, **num** 354-358
command, **patt** 359-364
command, **pattern** 365-368,506
command, **pc** 369-372
command, **pcset** 373-375
command, **perform** 1,376-380
command, **pf** 381-382
command, **pitch** 383-386
command, **proof** 387-388
command, **recode** 389-393
command, **record** 1,394-396
command, **regexp** 397-398
command, **reihe** 399-403
command, **rend** 404-407
command, **rid** 408-410,501
command, **scramble** 411-414
command, **semits** 415-417
command, **simil** 1,418-424
command, **smf** 1,425-426
command, **solfa** 427-430
command, **solfg** 431-433
command, **strophe** 215-216,434-440
command, **synco** 1,441-444
command, **tacet** 1,377,445
command, **thru** 188,210,215,219,436,438,446-450
command, **timebase** 98,451-452
command, **tonh** 453-456
command, **trans** 457-461
command, **urrhythm** 462-464
command, **veritas** 33,465-466
command, **vox** 467-468
command, **xdelta** 469-475
command, **yank** 210,476-480
command, **ydelta** 481-483
command, arguments 225
command, options 225
comments 9,303
comments, eliminate (**rid**) 408-410
comments, eliminate null (**rid**) 408-410
commission, reference records 31
commissioned works 26
common practice music, representation of (`**kern`) 96-106
common system barlines 43-45
compare two numerical spines (**correl**) 254
compare two numerical spines, (**correl**) 260
complementary character classes 498,504
composer's dates, reference records 29
composer's name 26
composer's name, reference records 28
composer's nationality, reference records 29
compound meter 34
concert program 189
concertina 195
concertino 187
concurrent events 9
concurrent numeric differences 481-483
conductor's name, reference records 30
consonance, representation of (`**diss`) 64-65
consonants, representation of (`**IPA`) 91
context dependency (`**kern`) 101
context, regular expressions 496,503
context command 245-253,315,320
contrabass 194
contrabassoon 195
contrafagotto 195
contralto voice 193
control-D 228,397
control-Z 228,397
controller card, reset MIDI (**midreset**) 344

Cope, D. 520
copyright date, reference records 32
copyright information 32-34
copyright message, reference records 32
copyright owner, reference records 32
copyright, country of 32
copyright, date 32
copyright, illegal 28
copyright, intellectual property 28
copyright, licensed 28
copyright, original document 32
copyright, original document owner 33
copyright, original year 33
copyright, owners 32
copyright, public domain 28
copyright, shareware 28,32
copyright, tampering with statements of 28
cor file extension 255
cor anglais 195
core pitch/duration representation (**kern) 96-106
cornemuse 195
cornet 195
cornett 195
cornetto 195
corporate composers, reference records 29
correl command 254-260
correl representation 49-50
correlation coefficient (**correl) 49-50
correlation, calculate (**correl**) 254-260
correlations, representation of (**correl) 49-50
counter-tenor course 77
counter-tenor voice 194
country of composition, reference records 31
country of copyright, reference records 32
courses, representation of 77
Croatian language 36,204
crossing hands 35
crumhorn — see instrument, krumhorn 196
curved lines 97
cut command (UNIX) 280
cut, see (**extract**)
cyclic rotation of row (**reihe**) 399-403
cymbals 197
cymbals, crash 197
cymbals, finger 197
cymbals, ride 197
cymbals, splash 197
Czech language 36,204
Czerny edition (**thru**) 447

D

d deletion (**humsed**) 310
Da Capos 188
Da Capos, expansion of (**thru**) 446-450
Dagomba dance 8
Dal Segnos 188
Damerau-Levenshtein metric 1,419,424
damping string (**fret) 78,81
dance scholars 7
Danish language 36,204
Dannenberg, R. 520
"Das Wandern" from *Die Schöne Müllerin* 24,434,436
data accuracy and error detection 521-522
data columns — see spines
data encoding (**encode**) 269-279
data records 9-10,303
data records, eliminate (**rid**) 408-410
data records, eliminate null (**rid**) 408-410
data records, number of 236
data tokens, numbering of (**num**) 354-358
date encoded, reference records 35
date of composition, reference records 31
date of first performance, reference records 30
date of first publication, reference records 32
date of performance, reference records 30
date, representation of (**date) 51-55
date representation 51-55
Dawkins, K. 1
dB, representation of (**dB) 56-57
dB representation 56-57
De Falla, M. 189
Debussy, C. 146
decibels, representation of (**dB) 56-57
decimal strophe numbers (**strophe**) 437
dedication, reference records 31
deep structure (**regexp**) 492
default fret tuning (**fret) 78
default key-velocity (**perform**) 379
default tempo (**perform**) 379
deg command 261-264
deg file extension 261
deg representation 58-60
degree, representation of scale (**deg) 58-60
degree, representation of scale (**degree) 61-63
degree, scale 261-268,427-430
degree command 265-268
degree representation 61-63
deletion (**d**), **humsed** 310
delimiters, subtoken (**cleave -d**) 241
delimiters, subtoken (**context -d**) 247
Delone, R. 520
Den Haag 31
Deng Xiao Tie 27
Density 21.5 248-249,251-252
density, determine textural 467-468
density, representation of textural (**vox#) 174-175
dgr file extension 265
diatonic interval 297-302,345-351
diatonic offset (**trans**) 457
diatonic pitch 261-268,427-430
diatonic transposition (**trans**) 457-461
Die Schöne Müllerin 24,434,436
dièse 431
diff command (UNIX) 102
differentiation 469-475
differing renditions (**strophe**) 434

diphthongs, representation of (**IPA) 91
direction of bowing (**fret) 79,81
direction of strumming (**fret) 79,81
"Dirge-Canons" from *In Memoriam Dylan Thomas* 402
diss command 1
diss representation 64-65
dissonance, representation of (**diss) 64-65
dissonance, sensory 1
do (solfège) 431
document modification description, reference records 33
dollars-sign in command line 230
Dorian mode 459
dotted staff line 214
double barlines 43
double barlines, ending without (**proof**) 387
double barlines, material following (**proof**) 387
double barlines, number of (**kern) 236
double barlines, representation of (**kern) 99
double barlines, with pauses 43
double beams, representation of (**kern) 97
double quotes in command line 230
double-flats, representation of (**kern) 96
double-flats, representation of **deg 264
double-flats, representation of **degree 268
double-sharps, representation of (**kern) 96
double-sharps, representation of **deg 264
double-sharps, representation of **degree 268
double-stops, create from separate spines 242
Dowling, G. 424,520
down-bows, representation of (**kern) 97
down-stems, representation of (**kern) 97
dramatic soprano voice 194
drum, bass 197
drum, snare 197
drum, tom-tom 197
dulcimer 194
dulcimer, tablature representation 76
duple meter 34
duplicate section labels (**thru**) 448
dur representation 66-68
duration names, American 98
duration tokens, incoherent (**proof**) 387
duration, eliminating fine details 451
duration, time-base 220
durationless notes, representation of (**kern) 98
durations, records spanning equivalent 451-452
durations, representation of (**dur) 66-68
durations, representation of (**kern) 97
durations, representation of (**recip) 134-136
durations, representation of dotted (**kern) 98
durations, representation of doubly-dotted (**kern) 98
Dutch language 36

E

Ecole Notre Dame 34
ed command (UNIX) 492
edition release date, reference records 32
editions, composite documents (**thru**) 446
editor, stream (**humsed**) 310-313
editorial footnote (**kern) 99
editorial interpretation (**kern) 99
editorial intervention (**kern) 99
editorial signifier, "*sic*" (**kern) 99
editorial signifier, *musica ficta* (**kern) 99
editorial signifier, invisible symbol (**kern) 99
egrep command (UNIX) 492
electric bass guitar 194
electric guitar 194
electric piano, Fender-Rhodes 197
electronic citation 35
electronic citation, example of 35
electronic documents, illegal 28
electronic edition 28
electronic edition version, reference records 33
electronic editor, reference records 33
electronic publisher checksum numbers 34-35,465-466
electronic publisher, reference records 32
elided phrases, representation of (**kern) 97
elided slurs, representation of (**kern) 97
eliminate duplicate barlines (**humsed**) 243
elision 97,101
Elliot, R. 523
embel representation 69-72
embellishments, 2-3 retardation 70
embellishments, 4-3 suspension 70
embellishments, 7-6 suspension 70
embellishments, 7-9 retardation 70
embellishments, 9-8 suspension 70
embellishments, accented lower neighbor 70
embellishments, accented passing tone 70
embellishments, accented upper neighbor 70
embellishments, anticipation 70
embellishments, appoggiatura 70
embellishments, arpeggio tone 70
embellishments, changing tone 70
embellishments, chromatic lower neighbor 70
embellishments, chromatic unaccented passing tone 70
embellishments, chromatic upper neighbor 70
embellishments, escape tone 70
embellishments, lower neighbor tone 70
embellishments, pedal point 70
embellishments, repeated tone 70
embellishments, representation of (**embel) 69-72
embellishments, retardation 70
embellishments, suspension 70
embellishments, unaccented passing tone 70
embellishments, upper neighbor tone 70
emotional states 8
empty files 304
encode command 1,269-279
encoder of document, reference records 33
encoding status, reference records 33
end delimiter (for extracting material) 476
end-of-file, control-D 228,397
end-of-file, control-Z 228,397
eng, representation of phonetic (**IPA) 92

english horn 195
English language 36,204
English language code 30
enhanced mode, Windows (**perform**) 379
ennunciation (`*ennun`) 162
ensemble 187
environment variable `AWK_VER` 398
era, representation of (`**Zeit`) 176-179
Erickson, R. 520
Es ist ein' Ros' entsprungen 481
ESC terminate **perform** 377,379
escape character () 230
escape character, regular expressions 493,503
escape tone (`**embel`) 70
esh, representation of phonetic (`**IPA`) 92
estimation of key (**key**) 328-331
Estonian language 36
eth, representation of phonetic (`**IPA`) 92
ethnomusicology 7
`ex` command (UNIX) 492
examples of regular expressions 499
exclamation marks 10
exclusive interpretations 10,303
exclusive interpretations, eliminate (**rid**) 408-410
expansion lists 188-189,219,436
expansion lists, definition of (**thru**) 447
expansion lists, duplicate labels (**thru**) 450
expansion lists, labelled (**thru**) 447
expansion lists, tandem interpretation 446
expansion lists, unlabelled (**thru**) 447
experimental stimulus 189
experimental subject 189
`expr` command (UNIX) 492
extended regular expression syntax 310
extended regular expressions 501,505
extended regular expressions, portability 311
extract -t, see **fields** command 286-289
extract specified bars (**yank**) 476-480
extract specified measures (**yank**) 476-480
extract specified passage (**yank**) 476-480
extract specified phrases (**yank**) 476-480
extract specified records (**yank**) 476-480
extract specified sections (**yank**) 476-480
extract strophic Humdrum data (**strophe**) 434-440
extract command 280-285
extract, field mode 280
extract, field-trace mode 280-281,286
extract, interpretation mode 280
extract, spine-path mode 280-281
extraction of parts (**extract**) 280-285

F

F clef 185
fa (solfège) 431
fagotto 195
falsetto voice 194
Fantini, D. 521
female voice 193
Fender-Rhodes electric piano 197
field mode (**extract**) 280
field-trace mode (**extract**) 280-281
fields command 286-289
fife 195
figured bass, not used in `**harm` 86
file extension `.cnt` 237
file extension `.cor` 255
file extension `.deg` 261
file extension `.dgr` 265
file extension `.frq` 294
file extension `.hmd` 339,394
file extension `.iv` 321,373
file extension `.krn` 324
file extension `.mac` 332
file extension `.met` 335
file extension `.mnt` 298,345
file extension `.nf` 352,373
file extension `.pc` 369
file extension `.pch` 453
file extension `.pcs` 373
file extension `.pf` 381
file extension `.pit` 383
file extension `.sem` 415
file extension `.sim` 419
file extension `.slg` 431
file extension `.sol` 428
file extension `.syn` 442
file extension `.tb` 452
file extension `.tr` 458
file extension `.urr` 463
file extension `.vox` 467
file number, reference records 33
file, placement of reference records 35
file-read (**r**), **humsed** 310
file-write (**w**), **humsed** 310,312
files, amalgamate (**assemble**) 232-234
files, empty 304
files, general information (**census**) 235-236
files, general information (**veritas**) 465-466
files, verification of 465-466
fill command 290-293
`find_reg.awk` file 480
`findpair.awk` file 480
fingering, index finger (`**fret`) 79,81
fingering, little finger (`**fret`) 79,81
fingering, middle finger (`**fret`) 79,81
fingering, representation of fret (`**fret`) 79,81
fingering, representation of plucking finger (`**fret`) 79,81
fingering, ring finger (`**fret`) 79,81
fingering, thumb (`**fret`) 79,81
finite state machine, implemented as (**record**) 396
Finnish language 36,204
first publisher, reference records 32
five-line staff 214
fixed template mode (**simil**) 418
Flemish language 36

Flottorp, G. 524
flugelhorn 196
flute; soprano, alto, bass 195
footfalls 11
form, fuga 34
form, minuet and trio 189
form, passacaglia 34
form, reference records 34
form, rondo 34,189
form, rounded binary 34,188
form, sonata-allegro 34,178,189
formants, vowels (**IPA) 94
format, abbreviated 188-189
format, through-composed 188-189
Forte, A. 124,321-322,352-353,373-375,381-382,520
fortepiano 194,197
forward string search (/), **perform** 377
four-line staff 185,214
frame, creation of contextual (**context**) 245
French language 36,204
French language code 30
French lute tablatures 76
French pitch notation (**solfg**) 431-433
French pitch, representation of (**solfg) 145-148
freq command 294-296
freq representation 73-75
frequency spectrum, representation of (**spect) 152-153
frequency, fundamental 294-296
frequency, representation of (**freq) 73-75
frequency, translate to (**freq**) 294-296
fret fingering, representation of (**fret) 79,81
fret position, representation of (**fret) 79,81
fret positions 76,81,190
fret representation 76-84
fretted tablatures, representation of (**fret) 76-84,190-191
frq file extension 294
Fuga 20 BWV 889b 13
fuga, form 34
full score, **assemble** 232-234
functional harmony, representation of (**harm) 85-90
fundamental frequency (**freq**) 294

G

g global substitution (**humsed**) 311
G clef 185
Gaelic language 36
generate numbers (**num**) 354-358
genre, barbershop quartet 34
genre, opera 34
genre, reference records 34
genre, slendro 34
genre, string quartet 34
German language 36,204
German language code 30
German lute tablatures 76
German pitch notation (**tonh**) 453-456
German pitch, representation of (**Tonh) 167-170
Ghahraman, D. 320,524
gittern 194
Glasberg, B. 522
global accidentals 202
global comments 9,303
global comments, eliminate (**rid**) 408-410
global comments, eliminate null (**rid**) 408-410
global comments, field-trace information 286
global substitution (**g**), **humsed** 311
Glockenspiel 197
gong 197
Gould, G. 188
grace notes, representation of (**kern) 98
great mean course 77
Greek language 36
green staff line 214
Greenwood, D. 520
grep command (UNIX) 26,34,397,492
gres command (UNIX) 492
groupettos, representation of (**kern) 98
Guido d'Arezzo 27
guitar 194
guitar, 12-string tablature representation 76
guitar, 6-string tablature representation 76
guitar, bass; tablature representation 76
guitar, riffs 7
Gute Nacht 31

H

Hall, P. 424,520
hammer-on articulation (**fret) 80-81
Hammond, organ 197
Handel, G.F. 7
Happy Birthday 12
harm representation 85-90
harmonic intervals 184
harmonic minor scale assumed (**deg**) 261
harmonic minor scale assumed (**degree**) 265
harmonic minor, use of (**harm) 85
harmonic number 192
harmonica 195
harmonics 192
harmonics, artificial (**fret) 78,81
harmonics, natural (**fret) 78,81
harmony, alternate (**harm) 87
harmony, bi-tonal (**harm) 87
harmony, explicit (**harm) 87
harmony, functional (**harm) 87
harmony, implicit (**harm) 87
harmony, key areas (**harm) 86
harmony, modulations (**harm) 86
harmony, polytonal (**harm) 87
harmony, representation of functional (**harm) 85-90

harmony, secondary dominants (`**harm`) 86-87
harp 194
harpsichord 194,197
Harris, M. 524
hautbois 196
Hebrew language 36
Heise, G.A. 522
Heldentenor voice 194
help screens, Humdrum 230
Hendrix, J. 7
hertz (**freq**) 294
heterophony 34
Hewlett, W. 2,520-521
hexadecimal number 445
hichiriki 196
Hill, M. 12
Hill, P. 12
Hindi language 36,204
hint command 297-302
historical musicology 7
`hmd` file extension 339,394
Hoboken catalogue 32
Holst, G. 67,135
honorifics, encoding of names 27
horn 195
Horwood, F. 521
Howard, R. 2
humdrum command 303-309
Humdrum commands 225
Humdrum, MIDI commands 225
Humdrum, asterisks 10
Humdrum, basic commands 225
Humdrum, carriage returns 10
Humdrum, comments 9
Humdrum, data encoding (**encode**) 269-279
Humdrum, data records 9-10
Humdrum, data tokens 11
Humdrum, exclamation marks 10
Humdrum, exclusive interpretations 10
Humdrum, general file information 235-236,465-466
Humdrum, global comments 9
Humdrum, interpretation keyword 11
Humdrum, interpretations 9
Humdrum, local comments 9
Humdrum, multiple-stops 13
Humdrum, newlines 10
Humdrum, null tokens 10-11
Humdrum, period character 10
Humdrum, record separator 12
Humdrum, records 9
Humdrum, specialized commands 225
Humdrum, spine paths 15
Humdrum, spine-path terminator 11
Humdrum, spines 10
Humdrum, subtoken 13
Humdrum, subtoken separator 13
Humdrum, syntax 8-26,303-309
Humdrum, tabs 10
Humdrum, tandem interpretations 10
Humdrum, token/spine separator 12
Humdrum, toolkit 8
Humdrum Toolkit version (**humver**) 314
Humdrum-extended regular expressions 501-502,505
humsed command 310-313,315,318
humver command 314
Hungarian Rhapsody No. 8 35
Hungarian language 36
hurdy-gurdy 194
Huron, D. 173,424,464,521,523-524
hypothesis testing 7

I

i insertion (**humsed**) 310
Icelandic language 36
idiophone instruments 199
illegal electronic documents 28
implied intervals, arising from multiple-stops 472
imprint information 32
index finger (`**fret`) 79,81
information measure (**infot**) 315-320
information theory (**infot**) 315-320
information, self-information (**infot**) 315
Indian tabla bols 8
infot command 315-320
inner intervals 473
inner intervals, elimination of 473
insertion (**i**), **humsed** 310
instrument class 199
instrument, 'cello 194
instrument, Bratsche 195
instrument, Fender-Rhodes electric piano 197
instrument, Glockenspiel 197
instrument, Irish bagpipe 195
instrument, Scottish bagpipe 195
instrument, ūd 195
instrument, accordion 195,197
instrument, archlute 194
instrument, bagpipe — French 195
instrument, bagpipe — Irish 195
instrument, bagpipe — Scottish 195
instrument, banjo 194
instrument, bassoon 195
instrument, bell 197
instrument, biwa 194
instrument, calliope 195
instrument, carillon 197
instrument, castanets 197
instrument, celesta 197
instrument, cembalo 194,197
instrument, ch'in 194
instrument, chalumeau 195
instrument, chimes 197
instrument, cittern 194
instrument, clarinet — piccolo, soprano, alto, bass 195
instrument, clavecin 194,197
instrument, clavichord 194,197

instrument, concertina 195
instrument, contrabass 194
instrument, contrabassoon 195
instrument, contrafagotto 195
instrument, cor anglais 195
instrument, cornemuse 195
instrument, cornet 195
instrument, cornett 195
instrument, cornetto 195
instrument, crumhorn — see instrument, krumhorn 196
instrument, cymbal — crash 197
instrument, cymbal — finger 197
instrument, cymbal — ride 197
instrument, cymbal — splash 197
instrument, cymbals 197
instrument, drum — bass 197
instrument, drum — snare 197
instrument, drum — tom-tom 197
instrument, dulcimer 194
instrument, electric bass guitar 194
instrument, electric guitar 194
instrument, electric piano — Fender-Rhodes 197
instrument, english horn 195
instrument, fagotto 195
instrument, fife 195
instrument, flugelhorn 196
instrument, flute — soprano, alto, bass 195
instrument, fortepiano 194,197
instrument, gittern 194
instrument, gong 197
instrument, guitar 194
instrument, harmonica 195
instrument, harp 194
instrument, harpsichord 194,197
instrument, hautbois 196
instrument, hichiriki 196
instrument, horn 195
instrument, hurdy-gurdy 194
instrument, kŏmun'go 194
instrument, kit 194
instrument, kokyū 194
instrument, koto 194
instrument, krumhorn — soprano, alto, tenor, bass 196
instrument, lute 194
instrument, mandolin 194
instrument, maracas 197
instrument, marimba 197
instrument, nōkan 196
instrument, oboe 196
instrument, oboe d'amore 196
instrument, ocarina 196
instrument, organ — Hammond 197
instrument, organ — pipe 196-197
instrument, organ — portative 196-197
instrument, organ — reed 196-197
instrument, panpipe 196
instrument, pianoforte 194,197
instrument, piatti 197
instrument, piccolo 196
instrument, pipa 194
instrument, pipe organ 196-197
instrument, portative organ 196-197
instrument, psaltery 194
instrument, qin 194
instrument, racket 196
instrument, rebec 194
instrument, recorder — sopranino, soprano, alto, tenor, bass 195
instrument, reed organ 196-197
instrument, sarod 194
instrument, sarrusophone 196
instrument, saxhorn — soprano, alto, tenor, baritone, bass, contrabass 196
instrument, saxophone — sopranino, soprano, alto, tenor, baritone, bass, contrabass 196
instrument, shakuhachi 196
instrument, shamisen 195
instrument, shawm — soprano, alto, tenor 195
instrument, sheng 196
instrument, sho 196
instrument, sitar 195
instrument, steel-drum 197
instrument, synthesizer 198
instrument, tablā 197
instrument, tambūrā 195
instrument, tambourine 197
instrument, tanbur 195
instrument, theorbo 195
instrument, timpani 197
instrument, tinpanny 197
instrument, triangle 197
instrument, trombone — tenor, bass 196
instrument, trumpet 196
instrument, tuba 196
instrument, ukulele 195
instrument, vînā 195
instrument, vibraphone 197
instrument, viola 195
instrument, viola d'amore 195
instrument, viola da gamba — treble, tenor, bass 195
instrument, violin 195
instrument, violoncello 194
instrument, xylophone 197
instrument, zūmā 196
instrument, zither 195
instrumentation 26
instrumentation, reference records 34
instruments 193-198
instruments, "voice" 199
instruments, brass 199
instruments, classification of 199
instruments, idiophone 199
instruments, keyboard 199
instruments, names of 193-198
instruments, number of 187
instruments, percussion 199
instruments, string 199
instruments, transposing 221
instruments, woodwind 199
integrate data types (**cleave**) 243

interface, MIDI card 269,376,394,396
International Phonetic Alphabet, representation of 91-95
interpretation keyword 11
interpretation mode (**extract**) 280
interpretation, tandem — `*ITr` 460
interpretation, tandem — `*I` 34
interpretation, tandem — `*Tr` 459-460
interpretation, tandem — `*[a-gA-G][-+]*:` 461
interpretation, tandem — `*k[ ]` 461
interpretation, tandem — key 461
interpretation, tandem — key signature 461
interpretation, tandem — transposition 459-460
interpretations 9,303
interpretations, eliminate (**rid**) 408-410
interpretations, eliminate exclusive (**rid**) 408-410
interpretations, eliminate null (**rid**) 408-410
interpretations, eliminate null tandem (**rid**) 408-410
interpretations, eliminate tandem (**rid**) 408-410
interpretations, number of 236
interval vector 373-375
interval vector (**iv**) 321-322
intervals, arising from multiple-stops 472
intervals, doubly-augmented (`**harm`) 86-87
intervals, doubly-diminished (`**harm`) 86-87
intervals, elimination of inner 473
intervals, harmonic 184
intervals, harmonic (**hint**) 297-302
intervals, harmonic (**ydelta**) 481-483
intervals, inner 473
intervals, melodic (**mint**) 345-351
intervals, melodic (**xdelta**) 469-475
intervals, permuted 473
intervals, permuted harmonic 184
intervals, pitch 297-302,321-322,345-351,469-475,481-483
Inutituuk language 12
Inventio 1 BWV 772 107
Inventio 8 BWV 779 256
inversion of row (**reihe**) 399-403
inverted mordents, representation of (`**fret`) 80-81
inverted mordents, representation of (`**kern`) 97
inverted mordents, semitone (`**fret`) 80-81
inverted mordents, whole tone (`**fret`) 80-81
inverted turns, representation of (`**fret`) 80-81
invisible staff line 214
invisible symbol, editorial signifier (`**kern`) 99
`ipa` file extension 92
`IPA` representation 91-95
irregular durations, representation of (`**kern`) 98
irregular meter indications 206
irregular meters 26,34
Ishizaki, M. 1
isolate data, **extract** 280
Italian language 29,36,204
Iltaliano, language 29
iv command 321-322
`iv` file extension 321,373

J

Japanese language 27,36,204
Japanese language code 30
Javanese language 36
Javanese-Indonesian names 27
Johnson-Laird, P. 1,171,173,462,464,522
joining sections (**thru**) 449
Josquin des Prez 27

K

Kameoka, A. 1,64-65,522
kern command 323-327
`kern` regular expressions, examples of 500
`kern` representation 96-106
Kernighan, B. 519-520
Kessler, E. 328,331,522
key context (`**harm`) 85
key context, undefine (`**harm`) 85
key indication, missing (**proof**) 387
key signatures, global accidentals 202
key signatures, missing (**proof**) 387
key signatures, pitch-class 202
key signatures, pitch-height 202
key signatures, precautionary 202-203
key signatures, tandem interpretation 15,202-203
key tandem interpretation 200-201
key, atonal 200
key, estimate of (**key**) 328-331
key, major 200
key, minor 200
key, tandem interpretation 461
key, undefined 200
key, unknown 200
key command 328-331
keyboard instruments 199
keyboard synthesizer 198
keys, tandem interpretation 15
keywords in character classes 499
Kipp, N. 523
Kirshenbaum, E. 91,95
kit 194
Klangfarbenmelodie 367
Klavierstück opus posthumous 168,400
Kliewer, V. 520
Kochan, S. 522
Köchel catalogue 32
kokyū 194
kŏmun'go 194
Korean language 36
koto 194
Krumhansl, C. 328,331,522
krumhorn; soprano, alto, tenor, bass 196
`krn` file extension 99,324
Kuriyagawa, M. 1,64-65,522

Kurkela, K. 522

L

Landowska, W. 188
la (solfège) 431
language code, English 30
language code, French 30
language code, German 30
language code, Japanese 30
language of original text, reference records 29
language text, representation of (`**text`) 160-164
language, Albanian 35
language, Arabic 35
language, Armenian 35
language, Azeri 35
language, Bengali 36,204
language, Bulgarian 36
language, Cambodian 36
language, Cantonese 36
language, Croatian 36,204
language, Czech 36,204
language, Danish 36,204
language, Dutch 36
language, English 36,204
language, Estonian 36
language, Finnish 36,204
language, Flemish 36
language, French 36,204
language, Gaelic 36
language, German 36,204
language, Greek 36
language, Hebrew 36
language, Hindi 36,204
language, Hungarian 36
language, Icelandic 36
language, Italian 36,204
language, Japanese 36,204
language, Javanese 36
language, Korean 36
language, Latin 36,204
language, Latvian 36
language, Lithuanian 36
language, Malayalam 36
language, Mandarin 36,204
language, Mongolian 36
language, Norwegian 36,204
language, Polish 36,205
language, Portugese 36,205
language, Romanian 36,205
language, Russian 36,205
language, Serbian 36,205
language, Slovak 36
language, Slovenian 36
language, Spanish 36,205
language, Swahili 36,205
language, Swedish 36,205
language, Tamil 36
language, Thai 36
language, Tibetan 36
language, Turkish 37
language, Ukranian 37
language, Urdu 37,205
language, Vietnamese 37
language, Welsh 37
language, Xhosa 37,205
language, Zulu 37,205
languages, tandem interpretation 204-205
last segment in input (**yank -r $**) 476
lateral vibrato (`**fret`) 80-81
Latin language 36,204
Latvian language 36
leading-tone, doubling of 7
leading-tone, encoding of (**deg**) 261
leading-tone, encoding of (**degree**) 265
Le sacre du printemps 30
Lee, C.S. 156,441,444,522
let ring (`**fret`) 78,81
Levelt, W.J.M. 523
Liberace 27
librarians, music 7
librettist, reference records 29
libretto, representation of (`**text`) 160-164
linear correlation (`**correl`) 49-50
link all pitches in a measure 245
Liszt, Franz 35
literals, regular expressions 492,503
Lithuanian language 36
local comments 9,303
local comments, eliminate (**rid**) 408-410
local comments, eliminate null (**rid**) 408-410
local comments, field-trace information 286
logical OR, regular expressions 497,504
longest match-max priority, regular expressions 496
Longuet-Higgins, H.C. 156,441,444,522
lower neighbor tones (`**embel`) 70
lowered roots (`**harm`) 85,87
lowered seventh degree (**deg**) 261
lowered seventh degree (**degree**) 265
lute 194
lute, French tablatures 76
lute, German tablatures 76
lute, tablature representation 76
lyric soprano voice 194
lyric tenor voice 194
lyricist, reference records 29
lyrics, representation of (`**text`) 160-164

M

m-dependency analysis (**infot**) 315
`mac` file extension 332
Madonna 27,29
Malayalam language 36

male voice 193
man command (UNIX) 231
Mandarin language 36,204
mandolin 194
mandoline, tablature representation 76,78,81
mandore, tablature representation 76
manual, on-line 231
manuscript acknowledgement, reference records 32
manuscript location, reference records 32
manuscript source, reference records 32
maracas 197
marimba 197
Martinů, B. 189
Mashinter, K. 1
m-dependency analysis (**infot**) 315
measure durations, incoherent (**proof**) 387
measure numbers 44
measure numbers, alternate 44
measure numbers, consistency of (**proof**) 387
measure, information (**infot**) 315-320
measure, link all pitches in 245
measures numbers, incrementing (**record**) 394
measures, extract specified (**yank**) 476-480
measures, numbering of (**num**) 354-358
measures, renumbering of (**num**) 354-358
mediant, encoding of (**solfa**) 427
medius finger (**fret) 79,81
melac command 332-333
melac representation 107-109
melodic accent (**melac**) 332-333
melodic accent, representation of (**melac) 107-109
melodic interval, representation of (**mint) 116-118
melodic, arch 7
Mendelssohn, Felix 27
met file extension 335
metacharacters in regular expressions 498,501
meter signature, tandem interpretation 15
meter signatures 206-207
meter signatures, missing (**proof**) 387
meter, irregular 34
meter, metric position (**metpos**) 334-338
meter, unannounced changes of (**proof**) 387
meters 206-207
meters, ametric 206
meters, irregular 206
meters, unknown 206
metpos command 334-338
metpos representation 110-112
metric accent, determine (**metpos**) 334-338
metric classification, reference records 34
metric hierarchy, (**urrhythm**) 462
metric position, determine (**metpos**) 334-338
metric position, representation of (**metpos) 110-112
metric syncopation 7
metric syncopation (**synco**) 441-444
metronome markings (**perform**) 379
metronome setting 217-218
mezzo soprano voice 193
mi (solfège) 431
Microsoft Windows (**perform**) 379
MIDI 339-344,376-380,394-396,425-426,445
MIDI card address 379,395,445
MIDI card, MPU-410 compatible 269,376,394,396
MIDI channel 208
MIDI channel (**midi**) 341
MIDI clock ticks (**record**) 395
midi command 1,339-343
MIDI default clock ticks (**midi**) 341
MIDI key velocity (**midi**) 342
MIDI multiple-stops (**midi**) 341
MIDI playback 376-377,394
MIDI quantizing (**record**) 395
MIDI synthesizer 376-377,394
MIDI tools 1
MIDI, all-notes-off 445
MIDI, controller card 344
MIDI, data encoding from (**encode**) 269-279
MIDI, format 0 files 426
MIDI, library functions 344,380,396,426,445
MIDI, performance 30
MIDI, record input 394-396
MIDI, representation of (**MIDI) 113-115
MIDI, reset 344,445
MIDI, sound output 376-380,425-426
MIDI representation 113-115
midreset command 344
Mikrokosmos No. 121 335
Miller, G.A. 522
min-max repetition, regular expressions 495,503
minimalist, style 34
minor scale — harmonic minor assumed (**deg**) 261
minor scale — harmonic minor assumed (**degree**) 265
minor, use of harmonic (**harm) 85
mint command 345-351
mint representation 116-118
minuet and trio 189
Miranda, E. 524
missing strophes 437
mnt file extension 298,345
mode insensitivity (****solfa**) 142
mode insensitivity (**solfa**) 427
mode, Aeolian 458
mode, Dorian 459
modification description, reference records 33
modulations (**harm) 86
Moles, A. 315,320,522
Mongolian language 36
Moore, B. 522
mordents ornament (**fret) 80-81
mordents, representation of (**kern) 97
mordents, representation of inverted (**kern) 97
mordents, semitone (**fret) 80-81
mordents, whole tone (**fret) 80-81
Mortice Kern Systems, Inc. 2,522
movement number, reference records 31
Mozart, W.A. 7
MPU-410 compatible MIDI card 269,376,394,396
multiple-stops 13
multiple-stops (rend) 406
multiple-stops (ydelta) 483

multiple-stops (**record**) 395
multiple-stops, MIDI (**midi**) 341
multiple-stops, create from separate spines 241-242
multiple-stops, create from successive records (**context**) 247
multiple-stops, implied intervals 472
multiple-stops, incoherent durations (**proof**) 387
multiple-stops, number of 236
Music Quest, Inc. 344,380,396,426,445
musica ficta, editorial signifier (`**kern`) 99
musicals 31
Muzak 29

N

Nettheim, N. 522
Newcomb, S. 522-523
Newcomb, V. 523
name, composer's 26
named strophe label 215-216
names, Javanese-Indonesian 27
names, aliases 27
names, anonymous 27
names, corporate 27
names, family name 27
names, honorifics 27
names, initials 27
names, of popular groups 27
names, stage names 27
nasals, representation of (`**IPA`) 91
nationality, reference records 29
natural harmonics (`**fret`) 78,81
naturals (`**kern`) 96
neighbor tones, lower (`**embel`) 70
neighbor tones, upper (`**embel`) 70
nested parentheses, regular expressions 504
nested phrases, representation of (`**kern`) 97
nested slurs, representation of (`**kern`) 97
newlines 10
next measure (+), **perform** 378
next measure (=), **perform** 378
nf command 352-353
`nf` file extension 352,373
no clef 185
nōkan 196
nominal duration, representation of (`**recip`) 134-136
non-Humdrum output (**thru**) 449
non-Humdrum output, possible (**extract**) 281,283
non-Humdrum output, problem of (**cleave**) 244
non-Western fretted instruments 76
non-chordal tones, representation of (`**embel`) 69-72
non-numeric strophe labels (**strophe**) 437
non-pitch-related signifiers (**tonh -x**) 456
Noorden, L. van 523
North Indian tabla bols 8
normal form, determine (**nf**) 352-353
Norwegian language 36,204
note, highest (`**kern`) 236
note, longest (`**kern`) 236
note, lowest (`**kern`) 236
note, reference records 35
note, shortest (`**kern`) 236
"note," prototype (**urrhythm**) 462
notes, number of (`**kern`) 236
notes, number of concurrent (`**kern`) 236
notes, numbering of (**num**) 354-358
notes, representation of (`**kern`) 96
null comments, eliminate (**rid**) 408-410
null data records, eliminate (**rid**) 408-410
null global comments, eliminate (**rid**) 408-410
null interpretations, eliminate (**rid**) 408-410
null interpretations, eliminate tandem (**rid**) 408-410
null local comments, eliminate (**rid**) 408-410
null records, eliminate data (**rid**) 408-410
null string as delimiter (**cleave**) 241
null tandem interpretations, eliminate (**rid**) 408-410
null tokens 10,303
null tokens, **fill** with data 290-293
null tokens, number of 236
null tokens, padding output with (**context**) 246
num command 354-358
number (**num**) 354-358
number of concurrent instruments 187
number, reference records 31
`number.awk` file 480
numerical differences 469-475,481-483
numerical strophe label 215-216

O

oboe 196
oboe d'amore 196
ocarina 196
octave number, encoding of (**degree**) 265
offset values (**xdelta**) 469
offset values, elimination of 469,473
on-line manual 231
one-line staff 214
open beams, representation of (`**kern`) 97
open strings, representation of 76,81
opera, genre 34
operas 31
Opolko, F. 523
opus number, reference records 31
OR, regular expressions 497,504
orchestrator, reference records 29
order, representation of sequential (`**ordo`) 119-120
ordinal range specification 476
`ordo` representation 119-120
organ, Hammond 197
organ, pipe 196-197
organ, portative 196-197
organ, reed 196-197
original copyright year, reference records 33

original document owner, reference records 33
original document, reference records 32
original language, reference information 27
ornaments, Wagnerian turns (**fret) 80-81
ornaments, incoherent (**proof**) 387
ornaments, inverted mordents (**fret) 80-81
ornaments, inverted turns (**fret) 80-81
ornaments, mordents (**fret) 80-81
ornaments, representation of (**kern) 96-97
ornaments, representation of generic (**kern) 97
ornaments, representation of inverted mordents (**kern) 97
ornaments, representation of inverted turns (**kern) 97
ornaments, representation of mordents (**kern) 97
ornaments, representation of trills (**kern) 97
ornaments, representation of turns (**kern) 97
ornaments, trills (**fret) 80-81
ornaments, turns (**fret) 80-81
Orpen, K. 1,424,523
Othello 34
orthographic information 96
ossia passages, *strophe 24,434
ossia strophe label 215
"other," prototype (**urrhythm**) 463
Ottman, R.W. 523
oud, tablature representation 76
Overill, R.E. 523
overlay 209
overlay, text 164,209

P

p panic MIDI reset (**perform**) 377
P power panic MIDI reset (**perform**) 377
panic (power) MIDI reset (**P**), **perform** 377
panic MIDI reset (**p**), **perform** 377
panpipe 196
parenthese (**fill**) 290
parentheses, regular expressions 494,504
Parncutt, R. 521,523
part extraction (**extract**) 280-285
partial beams, representation of (**kern) 97
passacaglia, form 34
Passacaglia on DSCH 27
passage, extract specified (**yank**) 476-480
passing tone, unaccented (**embel) 70
paste command (UNIX) 232
Pathetique Sonata 30
patt command 359-364,368
pattern matching (**patt**) 359-364
pattern matching (**pattern**) 365-368
pattern matching (**regexp**) 397-398
pattern, search for (**patt**) 359-364
pattern, search for (**pattern**) 365-368
pattern command 364-368,506
pauses, representation of (**kern) 97,99
pc command 369-372
pc file extension 369
pc representation 121-124
pch file extension 453
pcs file extension 373
pcset command 373-375
pcset representation 125-129
Pearson's correlation coefficient (**correl) 49-50
Pearson's correlation coefficient (**correl**) 254
pedal point (**embel) 70
pelog tuning 132
Perahia, M. 448
percussion instruments 199
perform command 1,376-380
perform terminate (**ESC**) 377
perform, proper termination of 379
performance information 30
performance practice 98
performance, different versions (**thru**) 446
performer's name, reference records 30
period of time, representation of (**Zeit) 176-179
period, Baroque 34
period, reference records 34
permuted intervals 473
pf command 381-382
pf file extension 381
pg command (UNIX) 492
phonemes, modifiers (**IPA) 91
phonemes, representation of (**IPA) 91
phonetics, consonants (**IPA) 91
phonetics, diphthongs (**IPA) 91
phonetics, eng (**IPA) 92
phonetics, esh (**IPA) 92
phonetics, eth (**IPA) 92
phonetics, formant (**IPA) 94
phonetics, nasals (**IPA) 91
phonetics, representation of (**IPA) 91-95
phonetics, schwa (**IPA) 92
phonetics, thorn (**IPA) 92
phonetics, vowels (**IPA) 91
phonetics, yogh (**IPA) 92
phrase marks, incoherent (**proof**) 387
phrases, extract specified (**yank**) 476-480
phrases, number of notes in (**num**) 357
phrases, numbering of (**num**) 354-358
phrases, pair first and last notes in 245
phrases, representation of (**kern) 96-97
phrases, representation of elided (**kern) 97,101
phrases, representation of nested (**kern) 97
piano fingerings 8
piano, Fender-Rhodes electric 197
pianoforte 194,197
piatti 197
piccolo 196
pick-up passages, extraction of 478
Pinying, Chinese transliteration 27,204
pipa 194
pipe organ 196-197
pipe, UNIX 229
pipeline, UNIX 229
pit file extension 383

pitch bending (**fret) 80-81
pitch tokens, incoherent (**proof**) 387
pitch, ANSI standard 383
pitch, French (**solfg**) 431-433
pitch, German (**tonh**) 453-456
pitch, **kern 96
pitch, **cents** 237-239
pitch, **deg** 261-264
pitch, **degree** 265-268
pitch, **freq** 294-296
pitch, **kern** 323-327
pitch, **midi** 339-343
pitch, **pcset** 373-375
pitch, **pitch** 383-386
pitch, **semits** 415-417
pitch, **solfa** 427-430
pitch, absolute 96
pitch, accent (**melac**) 332-333
pitch, bending of (**fret) 80-81
pitch, contour accent (**melac**) 332-333
pitch, diatonic (**deg**) 261-264
pitch, diatonic (**degree**) 265-268
pitch, diatonic (**solfa**) 427-430
pitch, interval 297-302,321-322,345-351,469-475,481-483
pitch, normal form 373-375
pitch, pitch-class **pc** 369-372
pitch, representation of (**pitch) 130-133
pitch, transposition (**trans**) 457-461
pitch-class 369-372
pitch-class set 373-375
pitch-class sets, representation of (**pcset) 125-129
pitch-class, sequential partitioning 245,252
pitch-classes, representation of (**pc) 121-124
pitch command 383-386
pitch representation 130-133
pitches, concurrent 297-302,481-483
pitches, determine number of concurrent 467-468
pitches, number of concurrent (**vox#) 174-175
pitches, sequential 345-351,469-475
pizzicato, plucked (**fret) 78,81
pizzicato, representation of (**kern) 97
place of first publication, reference records 32
place of performance, reference records 30
placement of reference records 35
The Planets 67,135
play MIDI 376
playback, MIDI 376-377,394
Plomp, R. 523
plucked pizzicato (**fret) 78,81
plucked ponticello (**fret) 78,81
plucked string (**fret) 78,81
plucked sul tasto (**fret) 78,81
plucked tremolo (**fret) 78,81
Polish language 36,205
pollex finger (**fret) 79,81
ponticello bowing (**fret) 78,81
ponticello, plucked (**fret) 78,81
popular title, reference records 30
portability, extended regular expressions 311
portative organ 196-197
Portugese language 36,205
position of staves within a system 213,215-216
POSIX standard 33,465
power panic MIDI reset (**P**), **perform** 377
Praetorius 481
Prather, R. 523
pre-defined representations 12
precautionary accidentals, missing (**proof**) 387
precision (**cents -p**) 237-238
precision (**correl -p**) 255
precision (**freq -p**) 294
precision (**semits -p**) 415
precision, default (**cents**) 238
precision, default (**correl**) 255
precision, default (**freq**) 295
precision, default (**semits**) 416
Prelude to *Tristan und Isolde* 117
prime form, determine (**pf**) 381-382
prime row (**reihe**) 399-403
Prime, T. 1
probabilities, long term (**infot**) 315
programming language 8
proof listening (**perform**) 376-380
proof command 387-388
psaltery 194
psychomusicology 7
publication status, reference records 32
publisher's catalogue number, reference records 32
publisher, reference records 32
pull-off articulation (**fret) 80-81

Q

qin 194
quadruple meter 34
quantizing, MIDI (**record**) 395
question mark, undefined value (**correl**) 260
question-mark in command line 230
quintuple meter 34
quintuplet durations, representation of (**kern) 98
quintus finger (**fret) 79,81
quotation marks in command line 230

R

r file-read (**humsed**) 310
Racinsky, T. 1
racket 196
Rahn, J. 124,322,353,375,382,523
raised roots (**harm) 85,87
raised seventh degree (**deg**) 261
raised seventh degree (**degree**) 265
randomize order of data records (**scramble**) 411-414

randomize order of data tokens (**scramble**) 411-414
rasgeado strumming (`**fret`) 79,81
Rastall, R. 523
ré (solfège) 431
rebec 194
`recip` representation 134-136
reciprocal duration, representation of (`**recip`) 98,134-136
recitativo voice 194
recode command 389-393
record repetition, regular expressions 502,505
record-count metacharacter 365-366
record command 1,394-396
recorder; sopranino, soprano, alto, tenor, bass 195
recording log 178
records, amalgamate (**context**) 245-253
records, congeal (**context**) 245
records, eliminate data (**rid**) 408-410
records, eliminate null data (**rid**) 408-410
records, extract specified (**yank**) 476-480
records, number of 236
records, number of data 236
records, randomize order (**scramble**) 411-414
records, spanning equivalent durations 451-452
red staff line 214
redundancy (**infot**) 318
reed organ 196-197
reference codes, several on a single record 27
reference information, files 26
reference information, original language 27
reference records 26-37
reference records, ASCII language setting 34
reference records, act number 31
reference records, alias/stage name 29
reference records, alternative title 30
reference records, analytic information 28,34
reference records, arranger 29
reference records, associated works 34
reference records, attributed composers 28
reference records, authorship information 28
reference records, checksum validation number 33,465-466
reference records, commission 31
reference records, composer's dates 29
reference records, composer's name 28
reference records, composer's nationality 29
reference records, conductor's name 30
reference records, copyright date 32
reference records, copyright information 28,32
reference records, copyright message 32
reference records, copyright owner 32
reference records, corporate composers 29
reference records, country of composition 31
reference records, country of copyright 32
reference records, date encoded 35
reference records, date of composition 31
reference records, date of first performance 30
reference records, date of first publication 32
reference records, date of performance 30
reference records, dedication 31
reference records, document modification description 33
reference records, edition release date 32
reference records, electronic edition version 33
reference records, electronic editor 33
reference records, encoder of document 33
reference records, encoding status 33
reference records, file number 33
reference records, first publisher 32
reference records, form 34
reference records, genre 34
reference records, imprint information 28,32
reference records, instrumentation 34
reference records, language of original text 29
reference records, librettist 29
reference records, lyricist 29
reference records, manuscript acknowledgement 32
reference records, manuscript location 32
reference records, manuscript source 32
reference records, metric classification 34
reference records, movement number 31
reference records, note 35
reference records, number 31
reference records, opus number 31
reference records, orchestrator 29
reference records, original copyright year 33
reference records, original document 32
reference records, original document owner 33
reference records, performance information 28,30
reference records, performer's name 30
reference records, period 34
reference records, place of first publication 32
reference records, place of performance 30
reference records, placement of 35
reference records, popular title 30
reference records, publication status 32
reference records, publisher 32
reference records, publisher's catalogue number 32
reference records, representation information 28,34
reference records, scene number 31
reference records, scholars' catalogue 32
reference records, style 34
reference records, suspected composer 28
reference records, suspected performer 30
reference records, title 30
reference records, title note 31
reference records, title of parent work 30
reference records, translated title 30
reference records, translator 29
reference records, user-defined signifiers 34
reference records, volume 31
reference records, warning 35
reference records, work identification information 28,30
refrains 188
refrains, through-composed expansions (**thru**) 446
regexp command 397-398
reggae, style 34
regular expression matching 359-368,397-398
regular expression tester (**regexp**) 397-398
regular expressions, Humdrum-extended 501-502,505
regular expressions, `**kern` 500

regular expressions, accidentals 500
regular expressions, anchors 496,503
regular expressions, barlines 500
regular expressions, basic 501,505
regular expressions, character classes 497-498,504
regular expressions, context 496,503
regular expressions, diatonic pitch 500
regular expressions, dotted durations 500
regular expressions, durations 500
regular expressions, escape character 493,503
regular expressions, examples of 499
regular expressions, extended 501,505
regular expressions, extended syntax 310
regular expressions, literals 492,503
regular expressions, logical OR 311,497,504
regular expressions, longest match priority 496
regular expressions, min-max repetition 495,503
regular expressions, nested parentheses 504
regular expressions, one-or-more 311
regular expressions, parentheses 494,504
regular expressions, phrases 500
regular expressions, record repetition 502,505
regular expressions, repetition operators 493,503
regular expressions, rests 500
regular expressions, simplifying 102
regular expressions, syntax 492
regular expressions, wild cards 493,503
regular expressions, zero-or-more 311
rehearsal order 189
reihe command 399-403
relative amplitude, representation of (`**dB`) 56-57
relative scale degree, representation of (`**deg`) 58-60
Renaissance period (`**Zeit`) 178
rend command 404-407
renumbering (**num**) 354-358
repeat measure (-), **perform** 378
repeat measure (_), **perform** 378
repeated measures, warning of (**proof**) 387
repeated passages 436
repeated signifiers (`**kern`) 101
repeated tone (`**embel`) 70
repeats, through-composed expansions (**thru**) 446
repetition operators, regular expressions 493,503
repetitions, expansion of (**thru**) 446-450
representation information 34-35
representation summary, (`**IPA`) 92
representation summary, (`**kern`) 99
representation, `**IPA` 91-95
representation, `**MIDI` 113-115
representation, `**Tonh` 167-170
representation, `**URrhythm` 171-173
representation, `**Zeit` 176-179
representation, `**cents` 46-48
representation, `**correl` 49-50
representation, `**dB` 56-57
representation, `**date` 51-55
representation, `**deg` 58-60
representation, `**degree` 61-63
representation, `**diss` 64-65
representation, `**dur` 66-68
representation, `**embel` 69-72
representation, `**freq` 73-75
representation, `**fret` 76-84,190-191
representation, `**harm` 85-90
representation, `**kern` 96-106
representation, `**melac` 107-109
representation, `**metpos` 110-112
representation, `**mint` 116-118
representation, `**ordo` 119-120
representation, `**pc` 121-124
representation, `**pcset` 125-129
representation, `**pitch` 130-133
representation, `**recip` 134-136
representation, `**semits` 137-139
representation, `**simil` 140-141
representation, `**simxrf` 419
representation, `**solfa` 142-144
representation, `**solfg` 145-148
representation, `**specC` 149-151
representation, `**spect` 152-153
representation, `**synco` 154-156
representation, `**takt` 157-159
representation, `**text` 160-164
representation, `**time` 165-166
representation, `**vox#` 174-175
representation, `fret` 190-191
representation, `hint` 184
representation, `recip` 206
representation, strophic (**strophe**) 434-440
representations, pre-defined 12
reset MIDI 445
reset MIDI card 344
rests, number of (`**kern`) 236
rests, numbering of (**num**) 354-358
rests, representation of (`**kern`) 98
retardation (`**embel`) 70
retardation, 2-3 (`**embel`) 70
retardation, 7-8 (`**embel`) 70
retrieving files 26
retrograde of row (**reihe**) 399-403
retrograde-inversion of row (**reihe**) 399-403
rhythmic prototype, representation of (`**URrhythm`) 171-173
rhythmic prototypes, characterize 462-464
Riemenschneider, A. 523
rid command 408-410,501
ripieno 187
Roland MPU-410 MIDI card 269,376,394,396
Roman alphabet 27
Roman numerals, use of (`**harm`) 85,87
Romanian language 36,205
Romanticism period (`**Zeit`) 178
rondo form 189
rondo, form 34
round-off (**cents**) 238
round-off (**freq**) 295
round-off (**pc**) 370
round-off (**semits**) 416
rounded binary form 188
rounded binary, form 34

row cyclic rotation (**reihe**) 399-403
row inversion (**reihe**) 399-403
row retrograde (**reihe**) 399-403
row retrograde-inversion (**reihe**) 399-403
row transposition (**reihe**) 399-403
Rowell, L. 520
ruber staff line 214
Russian language 29,36,205
Irusski, language 29

S

s substitute (**humsed**) 310
Le sacre du printemps 30
Salve Regina 7
sample filename `aeolian` 458
sample filename `bebop` 312
sample filename `berg` 375
sample filename `bigband` 312
sample filename `boulez` 375
sample filename `bwv779` 257
sample filename `chopin` 340
sample filename `density` 249
sample filename `dixieland` 312
sample filename `dorian` 459
sample filename `dvorak` 476
sample filename `faure` 479
sample filename `franck` 477
sample filename `fugue` 479
sample filename `ginastera` 477
sample filename `goldberg` 478
sample filename `guitar` 458
sample filename `haydn08` 478
sample filename `hendrix` 282
sample filename `hildegard` 280
sample filename `holst` 282
sample filename `igor` 402-403
sample filename `jellyroll` 311
sample filename `joplin` 477
sample filename `katsumi` 240
sample filename `koto` 357
sample filename `lassus` 280
sample filename `mahler` 478
sample filename `messiaen` 479
sample filename `milhaud` 458
sample filename `minuet` 479
sample filename `moonlight` 458
sample filename `mossolov` 282
sample filename `opus24` 322,353,382
sample filename `patrie` 391-392
sample filename `praetorius` 482
sample filename `ragtime` 311
sample filename `rebec` 358
sample filename `ricercar` 479
sample filename `sarod` 358
sample filename `sibelius` 282
sample filename `siegfrd.idl` 340
sample filename `sinfonia` 479
sample filename `song01` 405
sample filename `stamitz` 480
sample filename `swing` 312
sample filename `verdi` 477
sample filename `waltz` 479
sample filename `webern` 400-401,479
sample filename `wieck` 477
sample section label `Coda` 480
sample section label `First Theme` 478
sample section label `bridge` 478
sample section label `coda` 478
sample section label `exposition` 478
sample section label `minuet` 478
sample section label `second ending` 478
sample section label `trio` 478
Sandell, G. 1,524
sarod 194
sarrusophone 196
Sax, A. 178
saxhorn; soprano, alto, tenor, baritone, bass, contrabass 196
saxophone; sopranino, soprano, alto, tenor, baritone, bass, contrabass 196
scale degree, (**solfa**) 427-430
scale degree, absolute (**degree**) 265-268
scale degree, relative (**deg**) 261-264
scale degree, representation of (`**deg`) 58-60
scale degree, representation of (`**degree`) 61-63
scene number, reference records 31
scholars' catalogue, reference records 32
Schenkerian graphs 8
Schoenberg, A. 122
Schubert lyrics 7
Schubert, Franz 31
Schubert, Franz 24,31,434,436
Schumann, Robert 27
schwa, representation of phonetic (`**IPA`) 91-92
scordatura 190
scramble order of data records 411-414
scramble order of data tokens 411-414
scramble command 411-414
search for pattern (**patt**) 359-364
search for pattern (**pattern**) 365-368
search for pattern (**regexp**) 397-398
second derivative 469
second endings, through-composed expansions (**thru**) 446
section labels 188,210-211,219,436,478
section labels (**strophe**) 435
section labels, definition of (**thru**) 446
section labels, duplicate (**thru**) 448
section labels, extract according to 476
section labels, tandem interpretation 446
sections, definition of (**thru**) 446
sections, extract specified (**yank**) 476-480
sections, joining of (**thru**) 449
security, checksum validation number 34-35,465-466
`sed` command (UNIX) 310,492
Seewann, M. 524

segregate signifiers (**rend**) 404-407
self-information (**infot**) 315
Selfridge-Field, E. 2,520-521
sem file extension 415
semantic information 96
semitone mordents, representation of (**kern) 97
semitone trills, representation of (**kern) 97
semitones 415-417
semitones, representation of (**semits) 137-139
semits command 415-417
semits representation 137-139
sequential events 9
sequential numeric differences 469-475
sequential order, representation of (**ordo) 119-120
Serafini, S. 1
Serbian language 36,205
serial music 34
set variants (**reihe**) 399-403
seventh degree, lowered (**deg**) 261
seventh degree, lowered (**degree**) 265
seventh degree, raised (**deg**) 261
seventh degree, raised (**degree**) 265
seventh, added 70
Shakespeare, William 34
shakuhachi 196
shamisen 195
Shannon, C. 315,320
shawm; soprano, alto, tenor 195
shell metacharacters, problems in command line 281
shell variables 230
sheng 196
sho 196
si (solfège) 431
sic strophe label 215
sic, editorial signifier (**kern) 99
signifieds 8
signifiers 8
signifiers, canonical ordering of (**kern) 102-103
signifiers, segregate (**rend**) 404-407
signifiers, user-defined 34
silence — **tacet** command 445
silence signifier (**IPA) 91
sim file extension 419
simil.rc initialization file 419
simil command 1,418-424
simil representation 140-141
similarity, non-parametric (**simil**) 1,418-424
similarity, parametric (**correl**) 254-260
similarity, representation of (**correl) 49-50
similarity, representation of (**simil) 140-141
simple meter 34
Simpson, J. 1,173,464,524
simxrf representation 419
Sinfonia No. 13 for keyboard 350
single quotes in command line 230
single spine, encoding of (**encode**) 269
sitar 195
sitar, tablature representation 76
slendro tuning 132
slendro, genre 34
slg file extension 431
Slovak language 36
Slovenian language 36
slur marks, incoherent (**proof**) 387
slurs, representation of (**kern) 96-97
slurs, representation of elided (**kern) 97,101
slurs, representation of nested (**kern) 97
Smaill, A. 524
small mean course 77
smf command 1,425-426
snare drum 197
Social Sciences and Humanities Research Council 2
sol file extension 428
sol (solfège) 431
solfège 431-433
solfège, representation of (**solfg) 145-148
solfa syllables (**solfa**) 427-430
solfa syllables, representation of (**solfa) 142-144
solfa command 427-430
solfa representation 142-144
solfg command 431-433
solfg representation 145-148
solo 187
Sommermüd opus 48 (1933) 122
sonata-allegro form 34,189
soprano clef 185
soprano voice 193
sound recording log 178
source language, reference information 27
spaces, problem of trailing (**cleave**) 244
Spanish language 36,205
specC representation 149-151
spect representation 152-153
spectra, acoustical 8
spectra, instrument 1
spectral centroid, representation of (**specC) 149-151
spectrum, representation of frequency (**spect) 152-153
speed of strumming (**fret) 79,81
spiccato bowing (**fret) 78,81
spine organization 305
spine, split (**rend**) 404-407
spine-path changes (**extract**) 281,284
spine-path changes (**fill**) 290
spine-path indicators 212,303
spine-path mode (**extract**) 280-281
spine-path terminator 11
spines 10,212
spines, add (**cleave**) 240
spines, amalgamate (**assemble**) 232-234
spines, amalgamate (**cleave**) 240-244
spines, exchange (**cleave**) 240
spines, extraction (**extract**) 280-285
spines, join (**strophe**) 434
spines, spine-path structure (**fields**) 286-289
spines, split (**cleave**) 240
spines, split (**strophe**) 434
spines, strophe (**strophe**) 434
split spine (**rend**) 404-407
split tokens (**rend**) 404-407

square notation 8
staff number 213
staff, 0-line 214
staff, 1-line 214
staff, 4-line 185,214
staff, 5-line 214
staff, arrangement of lines 214
staff, colored lines 214
staff, dotted line 214
staff, invisible line 214
staff, number of lines 214
staff, position within a system 213
staff, red line 214
stage names 27
Stamitz, J. 189
standard mode, Windows (**perform**) 379
steel-drum 197
Steinweg, H. 178
stem-direction, representation of (`**kern`) 96-97
Stevens, S.S. 524
Stevenson, Ronald 27
stimulus, experimental 189
Stoll, G. 524
Stone, K. 524
Stradivari, A. 178
Straus, J. 124,322,353,375,382,524
Stravinsky, Igor 30,402
stream editor (**humsed**) 310-313
stress, indicators of syllable (`**IPA`) 91
string damping (`**fret`) 78,81
string instruments 199
string quartet, genre 34
string ringing (`**fret`) 78,81
strophe 215-216
strophe end indicator 215-216
strophe end indicator (`*S/fin`) 434,437
strophe interpretations on the same record 437
strophe interpretations, types of 437
strophe label, *ossia* 215
strophe label, *sic* 215
strophe label, named 215-216
strophe label, numerical 215-216
strophe labels, non-numeric (**strophe**) 437
strophe name label (`*S/`***number***) 437
strophe number label (`*S/`***number***) 437
strophe numbers, decimal (**strophe**) 437
strophe spines (**strophe**) 434
strophe, position within a system 215-216
strophe command 215-216,434-440
strophes, missing 437
strophic Humdrum data, extract (**strophe**) 434-440
strophic data 215
strophic passage initiator 215-216
strophic passage initiator (`*strophe`) 434,437
strophic passage terminator 215-216
strophic passage terminator (`*S-`) 434,437
structure, deep (**regexp**) 492
structure, surface (**regexp**) 492
strumming, direction of (`**fret`) 79,81
strumming, rasgeado (`**fret`) 79,81
strumming, speed of (`**fret`) 79,81
style, Ecole Notre Dame 34
style, bebop 34
style, minimalist 34
style, reference records 34
style, reggae 34
subject, experimental 189
submediant, encoding of (**deg**) 261
submediant, encoding of (**degree**) 265
substitute (s), **humsed** 310
substitute with matched string (&), **humsed** 312
subtoken 13
subtoken separator 13
sul tasto bowing (`**fret`) 78,81
sul tasto, plucked (`**fret`) 78,81
surface structure (**regexp**) 492
suspected composer, reference records 28
suspected performer, reference records 30
suspend playback (**<space>**), **perform** 377
suspension (`**embel`) 70
suspension, 4-3 (`**embel`) 70
suspension, 7-6 (`**embel`) 70
suspension, 9-8 (`**embel`) 70
suspensions 7
Swahili language 36,205
Swedish language 36,205
syllable, stress (`**IPA`) 91
synco command 1,441-444
`synco` representation 154-156
`syn` file extension 442
syncopation 462-464
syncopation, measurement of (**synco**) 441-444
syncopation, representation of (`**synco`) 154-156
"syncopation," prototype (**urrhythm**) 462
syntax checker, Humdrum (**humdrum**) 303-309
syntax checker, for `**kern` (**proof**) 387-388
synthesizer 198
synthesizer, MIDI 376-377,394
system, position of staves 213,215-216

T

tablā 197
tabla, North Indian bols 8
tablatures, absolute tuning 76,190
tablatures, relative tuning 76,190
tablatures, representation of fretted (`**fret`) 76-84,190-191
tabs 10
tacet command 1,377,445
take, recording log 178
`takt` representation 157-159
tambūrā 195
tambourine 197
tambours (`**fret`) 80-81
Tamil language 36
tampering with data, detection of 34,465-466

tanbur 195
tandem interpretation, *+ 212
tandem interpretation, *- 212
tandem interpretation, *> 188-189,210-211
tandem interpretation, *AT: 76,190
tandem interpretation, *Ch 208
tandem interpretation, *FT: 78,190
tandem interpretation, *H 192
tandem interpretation, *IC 199
tandem interpretation, *ICbras 199
tandem interpretation, *ICidio 199
tandem interpretation, *ICklav 199
tandem interpretation, *ICstr 199
tandem interpretation, *ICvox 199
tandem interpretation, *ICww 199
tandem interpretation, *ITr 221,460
tandem interpretation, *I 193-198
tandem interpretation, *K 202-203
tandem interpretation, *L 204-205
tandem interpretation, *MM 206-207,217-218
tandem interpretation, *RT: 77,190
tandem interpretation, *S- 24,216,434,437
tandem interpretation, *S/1 216
tandem interpretation, *S/*name* 437
tandem interpretation, *S/*number* 437
tandem interpretation, *S/fin 24,216,434,437
tandem interpretation, *S/name 216
tandem interpretation, *Tr 221,459-460
tandem interpretation, *[a-gA-g][-+]*: 461
tandem interpretation, *^ 212
tandem interpretation, *a2 187
tandem interpretation, *all 184
tandem interpretation, *clef 185-186
tandem interpretation, *conct 187
tandem interpretation, *ennun 162
tandem interpretation, *ensemb 187
tandem interpretation, *k[ ] 461
tandem interpretation, *k 202-203
tandem interpretation, *ripien 187
tandem interpretation, *solo 187
tandem interpretation, *staff 213
tandem interpretation, *strophe 24,215-216,434,437
tandem interpretation, *tb 220
tandem interpretation, *thru 219,448
tandem interpretation, *ueber 209
tandem interpretation, *unter 209
tandem interpretation, *v 212
tandem interpretation, *vibrato 162
tandem interpretation, *x 212
tandem interpretation, *|. 214
tandem interpretation, expansion lists 446
tandem interpretation, key 200-201,461
tandem interpretation, key signature 202-203,461
tandem interpretation, languages 204-205
tandem interpretation, section labels 446
tandem interpretation, transposition 459-460
tandem interpretations 10,303
tandem interpretations, eliminate (**rid**) 408-410
tandem interpretations, eliminate null (**rid**) 408-410
tb file extension 452
tee -a command (UNIX) 229
tee command (UNIX) 229
television schedules 8
Telugu notation 8
tempo 217-218
tempo indication, missing (**proof**) 387
tempo, changing (**perform**) 378-379
tempo, determining average (**midi**) 342
tempo, tandem interpretation 15
tenor clef 185
tenor course 77
tenor voice 193
tenore robusto voice 194
tenuto, representation of (**kern) 97
Terhardt, E. 524
terminate **perform (ESC)** 377
test file $HUMDRUM/regexp.txt 398
text, representation of (**text) 160-164
text representation 160-164
texture, heterophony 34
Thai language 36
theorbo 195
theorbo, tablature representation 76
third-note durations, representation of (**kern) 98
Thomas, R. 524
Thomassen, J. 332-333,524
Thomson, W. 520
thorn, representation of phonetic (**IPA) 92
through-composed data format 188-189,219,436
through-composed data format (**thru**) 446-450
"throw-away" representations 8
thru command 188,210,215,219,436,438,446-450
Tibetan language 36
tied notes (**deg**) 262
tied notes (**degree**) 266
tied notes (**freq**) 295
tied notes (**solfa**) 428
tied notes (**tonh -x**) 456
tied notes, incoherent (**proof**) 387
ties, representation of (**kern) 96-97
time signatures 206-207
time, constant (**timebase**) 451-452
time, representation of (**time) 165-166
time, representation of span of (**Zeit) 176-179
time-base 220
time-base (**encode**) 269
time-base, augmentation dots in 220
time-base, use of **recip durations 220
time representation 165-166
timebase command 98,451-452
timpani 197
tinpanny 197
title note, reference records 31
title of parent work, reference records 30
title, alternative 30
title, popular 30
title, reference records 30
tokens, amalgamate (**cleave**) 240-244
tokens, number of 236
tokens, number of multiple-stops in 236

tokens, number of null 236
tokens, numerical re-classification (**recode**) 389-393
tokens, randomize order (**scramble**) 411-414
tokens, split (**rend**) 404-407
tom-tom drum 197
tonal consonance, representation of (`**diss`) 64-65
tonal inflections in Chinese (`**IPA`) 91
tonality, bi-tonal harmony (`**harm`) 87
tonality, polytonal harmony (`**harm`) 87
tone row (**reihe**) 399-403
tonh command 453-456
`Tonh` representation 167-170
Tonhöhe, representation of (`**Tonh`) 167-170
tonic pitch, encoding of (**deg**) 261
tonic pitch, encoding of (**degree**) 265
tonic solfa syllables (**solfa**) 427-430
tonic solfa syllables, representation of (`**solfa`) 142-144
`tr` file extension 458
trailing spaces, problem of (**cleave**) 244
trans command 457-461
translated reference information 27
translated title, reference records 30
translator, reference records 29
transliteration (y), **humsed** 310
transliteration, Roman alphabet 27
transposing instruments 96,221,460
transposition 221
transposition (**trans**) 457-461
transposition of tone row (**reihe**) 399-403
transposition tandem interpretation 459-460
transposition, chromatic 221
transposition, chromatic (**trans**) 457-461
transposition, diatonic 221
transposition, diatonic (**trans**) 457-461
transverse vibrato (`**fret`) 80-81
treble clef 185
treble course 77
tremolo bowing (`**fret`) 78,81
tremolo, plucked (`**fret`) 78,81
triads, augmented (`**harm`) 85,87
triads, diminished (`**harm`) 85,87
triads, major (`**harm`) 85,87
triads, minor (`**harm`) 85,87
triangle 197
trills ornament (`**fret`) 80-81
trills, representation of (`**kern`) 97
trills, semitone (`**fret`) 80-81
trills, whole tone (`**fret`) 80-81
triple beams, representation of (`**kern`) 97
triple meter 34
triple-flats (`**kern`) 96
triple-sharps (`**kern`) 96
triplet durations, representation of (`**kern`) 98
Tristan und Isolde, Prelude to 117
trombone; tenor, bass 196
trumpet 196
tuba 196
tuning, absolute 76,190
tuning, pelog 132
tuning, quarter-tone fret positions (`**fret`) 78
tuning, relative 76,190
tuning, representation of cents deviation 76
tuning, representation of non-semitone fretting 77,190
tuning, slendro 132
tuning, vieil accord 77
Turkish language 37
turns ornament (`**fret`) 80-81
turns, representation of (`**kern`) 97
turns, representation of Wagnerian (`**kern`) 97
turns, representation of inverted (`**kern`) 97

U

ūd 195
ueber 209
Ukranian language 37
ukulele 195
unaccented passing tone (`**embel`) 70
undefine key context (`**harm`) 85
undefined key tandem interpretation 200
undefined value, question mark (**correl**) 260
underlay 209
underlay, text 164,209
unessential notes, representation of (`**embel`) 69-72
University of Illinois 320
UNIX `tee` 229
UNIX pipe 229
UNIX pipeline 229
unknown key tandem interpretation 200
unknown meter indications 206
unmatched data (**correl**) 256
unpublished works 32
unter 209
up-bows, representation of (`**kern`) 97
up-stems, representation of (`**kern`) 97
upper neighbor tone (`**embel`) 70
`urr` file extension 463
Ur-rhythm theory (**urrhythm**) 462-464
urrhythm command 462-464
`URrhythm` representation 171-173
Urdu folk songs 7
Urdu language 37,205
user-defined signifiers, reference records 34

V

van Noorden, L. 523
van Vianen, B. 524
Varèse, Edgar 248-249,251-252
variable template mode (**simil**) 419
vector, interval (**iv**) 321-322
verification of files 465-466
veritas command 33,465-466

verses 188
verses (**strophe**) 434-440
verses, order of (**strophe**) 434
verses, strophic data 215
version labels, duplicate (**thru**) 450
version of Humdrum Toolkit (**humver**) 314
version of electronic edition 33
version, definition of (**thru**) 447
versions 188
versions, expansion of (**thru**) 446-450
vi command (UNIX) 492
Vianen, B. van 524
vibraphone 197
vibrato (*vibrato) 162
vibrato, lateral (**fret) 80-81
vibrato, transverse (**fret) 80-81
Victoria, Tomas Luis de 7
Vietnamese language 37
vīnā 195
viol, tablature representation 76
viola 195
viola d'amore 195
viola da gamba; treble, tenor, bass 195
violin 195
violoncello 194
viridis staff line 214
voice, Heldentenor 194
voice, alto 194
voice, baritone 193
voice, bass 193
voice, basso cantante 194
voice, basso profondo 194
voice, castrato 194
voice, child's 193
voice, coloratura soprano 194
voice, contralto 193
voice, counter-tenor 194
voice, dramatic soprano 194
voice, falsetto 194
voice, female 193
voice, generic 193
voice, lyric soprano 194
voice, lyric tenor 194
voice, male 193
voice, mezzo soprano 193
voice, recitativo 194
voice, soprano 193
voice, tenor 193
voice, tenore robusto 194
voices 193-198
voices (*ICvox) 199
voices, names of 193-198
Voiles 146
volume, reference records 31
Vos, J. 524
vowel coloration 7
vowels, formants (**IPA) 94
vowels, representation of (**IPA) 91
vox# representation 174-175
vox command 467-468
vox file extension 467
w file-write (**humsed**) 310,312

W

Wagner, R. 8,117
Wagnerian turns, representation of (**fret) 80-81
Wagnerian turns, representation of (**kern) 97
"Das Wandern" from *Die Schöne Müllerin* 24,434,436
Wapnick, J. 523
warning, reference records 35
warnings, suppression of (**proof**) 387
Waterloo 2
Weaver, W. 320
Webern, A. 168
Webern, Anton von 400
Weinberger, P. 519-520
Welsh language 37
whole-tone mordents, representation of (**kern) 97
whole-tone trills, representation of (**kern) 97
Wiggens, G. 524
wild cards, regular expressions 493,503
Windows (**perform**) 379
Windows enhanced mode (**perform**) 379
Windows standard mode (**perform**) 379
Winterreise 31
Wong, A. 320,524
Wood, P. 522
woodwind instruments 199
work identification information 30-31

X

xdelta command 469-475
Xhosa language 37,205
x_option.awk file 239,264,268,296,327,372,385,417,430,433,456
xylophone 197

Y

y transliteration (**humsed**) 310
yank command 210,368,476-480
Yates, J. 524
ydelta command 481-483
yogh, representation of phonetic (**IPA) 92

Z

Zeit representation 176-179
zero-line staff 214
zither 195
Zulu language 37,205
zūrnā 196
Zwicker, E. 524

ISBN 0-936943-10-6