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)