Chapter 13

Assembling Scores

In the previous chapter we learned how to extract parts and passages from Humdrum files. In this chapter we discuss the reverse procedures: how to assemble and coordinate larger documents from individual segments and parts. We will discuss four tools: the UNIX **cat** command, and the Humdrum **assemble**, **timebase** and **rid** commands.

The *cat* Command

The UNIX **cat** command allows two or more inputs to be concatenated together. If we concatenate two files, the output will consist of the contents of the first file, followed immediately by the contents of the second file. For example, the following command will concatenate together the files `mov.1`, `mov.2` and `mov.3` and place the result in the file `complete`. The order of concatenation is the same as the order of file names given on the command line:

```
cat mov.1 mov.2 mov.3 > complete
```

If each of the concatenated files conforms to the Humdrum syntax, then the resulting combined file is guaranteed to conform to the Humdrum syntax. However, in many cases, there may be redundant information present in the concatenated output. Suppose we had three files, each of which encoded a single measure of music. Concatenating them together might result in an output such as the following:

```
!! File 1
**kern **kern **kern **kern
*M4/4  *M4/4  *M4/4  *M4/4
*G:    *G:    *G:    *G:
*k[f#] *k[f#] *k[f#] *k[f#]
4GG    4B     4d     4g
=1     =1     =1     =1
4GG    4d     4g     4b
4FF#   4d     [4a    4dd
8GG    4d     8a]    4b
8BB    .      [8g    .
4D     4d     8g]    4a
```

```
.       .       8f#     .
*-      *-      *-      *-
!! File 2
**kern  **kern  **kern  **kern
*M4/4   *M4/4   *M4/4   *M4/4
*k[f#]  *k[f#]  *k[f#]  *k[f#]
=2      =2      =2      =2
8G      4d      4g      4b
8F#     .       .       .
4E      4e      4g      4cc#
4D;     4A;     4f#;    4dd;
8A      4cn     4a      4ee
8G      .       .       .
*-      *-      *-      *-
!! File 3
**kern  **kern  **kern  **kern
*M4/4   *M4/4   *M4/4   *M4/4
*k[f#]  *k[f#]  *k[f#]  *k[f#]
*k[f#]  *k[f#]  *k[f#]  *k[f#]
=3      =3      =3      =3
4F#     4d      [4a     8dd
.       .       .       8cc
4G      4d      8a]     4b
.       .       [8g     .
8C#     8e      8g]     4a
8D#     4B      8f#     .
8E      .       8e      8g
8F#     8A      8d      8a
*-      *-      *-      *-
```

Notice that each complete measure ends with spine-path terminators and that the `**kern` exclusive interpretations are repeated. This organization has a number of repercussions for various Humdrum tools. For example, the **mint** command calculates melodic intervals between successive notes within a spine. However, **mint** will not calculate intervals between pitches that are separated by a spine-path terminator. In other words, in the above output, **mint** will fail to calculate the melodic intervals between notes in successive measures.

The *rid* Command

This problem can be resolved by using the Humdrum **rid** command. The **rid** command can be used to eliminate various kinds of records. Each option for **rid** eliminates a different class of records. Here are the record classes with their associated options:

-G eliminate all global comments
-g eliminate only global comments that are empty
-L eliminate all local comments
-l eliminate only local comments that are empty

-I eliminate all interpretations
-i eliminate only null interpretations
-D eliminate all data records
-d eliminate only null data records
-T eliminate all tandem interpretations
-t eliminate duplicate (repeated) tandem interpretations
-U eliminate unnecessary exclusive interpretations (see below)
-u same as -U

Null records are devoid of content. For example, null interpretations consist of a single asterisk in each spine; null data record consists of just null data tokens (.) in each spine; null local comments consist of a single exclamation mark in each spine. Null global comments contain just two exclamation marks at the beginning of a record.

Upper- and lower-case options are used to distinguish *all* records of a certain class (upper-case) from *empty*, *null* or *repeated* records of a certain class (lower-case).

By way of example, the following command will eliminate all global comments (including reference records) from the input:

```
rid -G Saint-Saens
```

Similarly, the following command will eliminate all tandem interpretations from an input:

```
rid -T Vaughan-Williams
```

Options can be combined. The following command eliminates all global and local comments, interpretations, and null data records:

```
rid -GLId
```

The option combination **-GLId** is very common with **rid** since only non-null data records are retained in the output.

With the **-u** option, **rid** will remove "unnecessary" exclusive interpretations. Exclusive interpretations are deemed unnecessary if they don't change the current status of the data. In the following example, the second `**psaltery` interpretation is redundant. The **rid -u** command would remove the first spine-path terminator and the second exclusive interpretation — leaving a continuous data spine.

```
**psaltery
.
.
.
*-
**psaltery
.
*-
```

In addition, **rid** provides a **-t** option which removes "duplicate" or repeated tandem interpretations. In the above example there is no need to repeat the meter signature and key signature in each measure. The following command will concatenate each of the three measures together, and then eliminate the unwanted interpretations:

```
cat bar1 bar2 bar3 | rid -ut
```

The resulting output is given below:

```
!! File 1
**kern  **kern  **kern  **kern
*M4/4   *M4/4   *M4/4   *M4/4
*k[f#]  *k[f#]  *k[f#]  *k[f#]
4GG     4B      4d      4g
=1      =1      =1      =1
4GG     4d      4g      4b
4FF#    4d      [4a     4dd
8GG     4d      8a]     4b
8BB     .       [8g     .
4D      4d      8g]     4a
.       .       8f#     .
!! File 2
=2      =2      =2      =2
8G      4d      4g      4b
8F#     .       .       .
4E      4e      4g      4cc#
4D;     4A;     4f#;    4dd;
8A      4cn     4a      4ee
8G      .       .       .
!! File 3
=3      =3      =3      =3
4F#     4d      [4a     8dd
.       .       .       8cc
4G      4d      8a]     4b
.       .       [8g     .
8C#     8e      8g]     4a
8D#     4B      8f#     .
8E      .       8e      8g
8F#     8A      8d      8a
*-      *-      *-      *-
```

Of course care should be exercised when concatenating inputs together. Although an output may conform to the Humdrum syntax, the result can nevertheless violate conventions for a specific representation such as `**kern`. For example, if we were to concatenate measure 85 to measure 87, it is possible that tied-notes won't match up, or that phrases will begin without ending, etc. These anomalies may cause problems with subsequent processing.

Assembling Parts Using the *assemble* Command

Assembling parts into a full score is slightly more challenging than concatenating together musical segments. The principle tool for joining spines together is the **assemble** command. Consider the following two files:

```
!! Assemble example
!! File 1
**Letters
! A to E
A
B
C
D
E
*-
```

```
!! Assemble example
!! File 2
**Numbers
! 1 to 5
1
2
3
4
5
*-
```

These two files can be aligned side by side using **assemble**:

```
assemble letters numbers
```

The resulting output is:

```
!! Assemble example
!! File 1
!! File 2
**Letters **Numbers
! A to E  ! 1 to 5
A         1
B         2
C         3
D         4
E         5
*-        *-
```

Note the following: (1) The spines are joined side by side from left to right in the same order as specified on the command line. (2) Local comments are preserved in their appropriate spines. (3) When identical global comments occur at the same location in both files, only a single instance of

the comment is output. (4) Dissimilar global comments are output successively.

The files joined by **assemble** are not confined to a single spine. For example, one input file may contain 2 spines and a second file may contain 20 spines. The resulting output will contain 22 spines. There is no limit to the number of files that can be assembled at one time.

In many cases, the input files will have dissimilar lengths. The **assemble** command will correctly terminate the appropriate spines. For example, in the above case, if the `numbers` file contained only the numbers 1 to 3, the assembled output would appear as follows:

```
!! Assemble example
!! File 1
!! File 2
**Letters **Numbers
! A to E  ! 1 to 3
A         1
B         2
C         3
*         *-
D
E
*-
```

If the order of the input files was reversed, **assemble** would produce an output with the appropriate spine-path changes:

```
!! Assemble example
!! File 2
!! File 1
**Numbers   **Letters
! 1 to 3    ! A to E
1           A
2           B
3           C
*-          *
D
E
*-
```

Note that if all of the input files conform to the Humdrum syntax, then **assemble** guarantees that the assembled output will also conform to the Humdrum syntax.

Aligning Durations Using the *timebase* Command

Suppose now that we wanted to join two hypothetical files containing `**kern` data. The first file contains two quarter notes, whereas the second file contains four eighth notes:

```
!! File 1
**kern
4c
4d
*-

!! File 2
**kern
8e
8g
8f
8g
*-
```

Using **assemble** to paste them together will clearly lead to an uncoordinated result. The two quarter notes in file 1 will be incorrectly matched with the first two eighth notes in file 2.

The Humdrum **timebase** command can be used to reformat either `**kern` or `**recip` inputs so that each output data record represents an equivalent slice (elapsed duration) of time. (Barlines are ignored by **timebase**.) The **timebase** command achieves this by padding an input with null data records. In the above case, we would preprocess file 1 as follows:

```
timebase -t 8 file1 > file1.tb
```

The new file would look like this:

```
!! File 1
**kern
4c
.
4d
.
*-
```

The **-t** option is used to indicate the "time base" — in this case, an eighth duration. Since all non-barline data records in both files represent elapsed durations of an eighth-note, we can continue by using the **assemble** command as before. The command:

```
assemble file1.tb file2
```

will result in the following two-part score:

```
!! File 1
!! File 2
**kern      **kern
4c          8e
.           8g
4d          8f
```

```
.           8g
*-          *-
```

Suppose that `file2` also contained a quarter-note. For example, consider a revised `file2`:

```
!! File 2
**kern
8e
8g
4f
*-
```

Before assembling the two parts, we would need to apply the **timebase** command to this file (using the same 8th-note time-base value). Assembling the two "time-based" files would produce the following result:

```
!! File 1
!! File 2
**kern      **kern
4c          8e
.           8g
4d          4f
.           .
*-          *-
```

Notice that we have a spurious null data record in the last line; both parts encode null tokens. For most processing, the presence of null data records is inconsequential. However, if we wish, these null data records can be eliminated using the **rid** command with the **-d** option. In fact it is common to follow an **assemble** command with **rid -d** to strip away unnecessary null data records. The command:

```
assemble file1.tb file2.tb | rid -d
```

would result in the following output:

```
!! File 1
!! File 2
**kern      **kern
4c          8e
.           8g
4d          4f
*-          *-
```

The **timebase** command can be applied to multi-spine inputs as well as single-spine inputs. Consider, the following input:

```
**kern   **kern **kern **kern **commentary
4g       8.r    8.cc   16ee   2nd inversion
.        .      .      8ff    .
.        32b    16cc   16gg   clash
.        32a    .      .      .
8f       8cc    8dd    8ff    suspension
*-       *-     *-     *-     *-
```

The following command will cause the addition of null data records so that each data record represents an elapsed time of a 32nd duration. Incidentally, notice that any spine contain non-rhythmic data — such as the `**commentary` spine in the above example — is also transformed so that synchronous data is maintained.

```
timebase -t 32 Corelli
```

The corresponding output is as follows.

```
**kern   **kern **kern **kern **commentary
*tb32    *tb32  *tb32  *tb32  *tb32
4g       8.r    8.cc   16ee   2nd inversion
.        .      .      .      .
.        .      .      8ff    .
.        .      .      .      .
.        .      .      .      .
.        .      .      .      .
.        32b    16cc   16gg   clash
.        32a    .      .      .
8f       8cc    8dd    8ff    suspension
.        .      .      .      .
.        .      .      .      .
.        .      .      .      .
*-       *-     *-     *-     *-
```

Notice that **timebase** has added a tandem interpretation (`*tb32`). This indicates that the output has been processed so that each non-barline data record represents an elapsed duration equivalent to a thirty-second note.

Assembling N-tuplets

Typically, one can simply use the shortest duration present as a guide for a suitable time-base value. The shortest duration can be determined using the **census -k** command described in Chapter 4. However, tuplets require a little more sophistication. Suppose we wanted to assemble two parts, one containing just eighth-notes and the other containing just quarter-note triplets. (The quarter-note triplets will be encoded as three notes in the time of a half-note, or "6th" notes.) We need to create an output whose rhythmic structure will appear as follows:

```
**kern    **kern
*M2/4     *M2/4
=1        =1
6         8
.         8
6         .
.         8
6         .
.         8
=2        =2
6         8
.         8
6         .
.         8
6         .
.         8
=3        =3
*-        *-
```

In this case, choosing a time-base according to the shortest duration (8th) will not work since a 6th note is not an integral multiple of the eighth duration. We need to find a *common duration factor* for both values. The shortest common duration is a 24th note (there are three 24th notes in an 8th note, and four 24th notes in a 6th note). Applying the time-base value '24' to both files will allow us to coordinate them properly. Remember that **rid -d** can be used to eliminate unnecessary null data records once we have finished assembling the spines. In the worst case, you can determine a common duration factor by simply multiplying together the shortest notes in the files to be assembled. For example, $6 \times 8 = 48$; so a time-base of 48 will be guaranteed to work for both files.

Checking an Assembled Score Using *proof*

In assembling any score from a set of parts, there is always the danger of using the wrong time-base value. When parts are miscoordinated, it is typically the consequence of one or more notes being discarded by **timebase**. Fortunately, such miscoordinations are easily detected by applying the **proof** command to any assembled `**kern` output. The **proof** utility checks `**kern` representations for a wide variety of possible encoding errors or ambiguities:

```
proof fullscore
```

By way of summary, creating a full score from a set of `**kern` parts involves the following five tasks: (1) Identify a common duration factor for all the parts. Use **census** to determine the shortest duration; if any of the parts contains an N-tuplet, then the common duration factor may be smaller than the shortest note. (2) Use the **timebase** command to expand each input file separately using the common duration factor. (3) Assemble the parts using **assemble**. (4) If desired, eliminate unnecessary null data records using **rid -d**. (5) Check the assembled score for rhythmic coherence using the **proof** command.

Other Uses for the *timebase* Command

The most common use of **timebase** is as a way of expanding a file by padding it with null data records. However, **timebase** can also be used to contract a file, giving us only those sonorities that lie a fixed duration apart. For example, specifying a time-base of **-t 2** will cause only those sonorities that are separated by a half-note duration to be output. This sort of rhythmic reduction can be useful in certain circumstances. For example, suppose you suspect there may be a hemiola tendency in a given work by Brahms, where the duration separating hemiola notes is a dotted-quarter. The command:

```
timebase -t 4. brahms
```

can be used to extract only those sonorities that are separated by a dotted-quarter duration.

Similarly, suppose we want to extract all sonorities falling on the third beat of a waltz written in 3/2 meter. First we would edit the input file so it begins on the third beat of some measure. Then we could use the following command:

```
grep -v ^= waltz | timebase -t 1. > 3rd_beat
```

Note that the use of **grep** here is essential in order to eliminate barlines. The **timebase** command resets itself with each barline, so time-base durations are calculated from the beginning of the bar. When barlines are eliminated, **timebase** cannot synchronize to the beginning of each bar and so simply floats along at the fixed time-base.

Additional Uses of *assemble* and *timebase*

Although we normally assemble parts together, sometimes it is useful to assemble entire scores together. Suppose we wanted to listen to a theme at the same time as one of its variations. We might first use **yank** to extract the appropriate sections. At the same time we might determine a common duration factor and expand them using **timebase**.

```
yank -s Theme -r 1 blacksmith | timebase -t 32 > temp1
yank -s 'Variation 1' -r 1 blacksmith | timebase -t 32 > temp2
```

Then we assemble the two sections together, translate to the `**MIDI` representation and use **perform** to listen to both sections at the same time:

```
assemble temp1 temp2 | midi | perform
```

Similarly, suppose we would like to compare the bass lines for each variation in some set. We might extract each of the bass lines, assemble them into a single score, and then use the **ms** and **ghostview** commands to allow us to see all of the bass lines for all of the variations concurrently.

```
yank -s 'Variation 1' -r 1 goldberg | timebase -t 16 > temp1
yank -s 'Variation 2' -r 1 goldberg | timebase -t 16 > temp2
  etc. ...
assemble temp1 temp2 temp3 ... | rid -d | ms > basslines.ps
```

```
ghostview basslines.ps
```

The most common use of **assemble** is not to assemble parts, but to assemble different types of concurrent information. Suppose we would like to determine whether descending minor seconds are more likely to be *fah-mi* rather than *doh-ti*. We can use the **mint** command to characterize melodic intervals, and the **solfa** command to characterize scale degrees. Assume that our input is monophonic:

```
mint melodies > temp1
solfa melodies > temp2
```

The files `temp1` and `temp2` will have the same length, so we can assemble them together. This will generate an output consisting of two spines, `**mint` and `**solfa`. In effect, the `**mint` spine data will tell us the interval used to approach the scale degree encoded in the `**solfa` spine. We can use **grep** to search for the appropriate combinations of interval and scale degree and count the number of occurrences:

```
assemble temp1 temp2 | grep -c '-m2.*mi'
assemble temp1 temp2 | grep -c '-m2.*ti'
```

This same approach can be used to address (innumerable) questions pertaining to concurrent patterns. For example, suppose we have a `**harm` spine that identifies the 'Roman numeral' functional harmony for some choral work. We can identify complex situations such as the following: for the soprano voice, count how many subdominant pitches are approached by an interval of a rising third or a rising sixth and coincide with a dominant seventh chord. First, let's extract the soprano line and create a corresponding scale degree representation using **deg**. We can use the **-a** option to avoid outputting the melodic direction signifiers (^ and v):

```
extract -i '*Isopran' howells | deg -a > temp1
```

Next, let's again extract the soprano voice and create a corresponding melodic interval representation using **mint**. Since we are not interested in interval qualities we can invoke the **-d** option to output only diatonic interval sizes.

```
extract -i '*Isopran' howells | mint -d > temp2
extract -i '**harm' howells > temp3
```

We have also extracted the `**harm` spine and placed it in the file `temp3`. If we assemble together our three temporary files, the result will have three spines: `**deg`, `**mint` and `**harm`. We can now use **grep** to search and count all instances of subdominant pitches that are approached by ascending thirds/sixths and that coincide with dominant seventh chords (in the `**kern` representation: 'V7'):

```
assemble temp1 temp2 temp3 | grep -c '^4<tab>+[36]<tab>V7
```

The **timebase** command can also be used for tasks other than assembling parts together. Suppose we would like to determine whether secondary dominant chords are more likely to appear on the third beat than other beats in a triple meter work. The **timebase** command can be used to reformat a score so that each measure occupies the same number of data records. For example, in a 3/4 me-

ter, an eighth-note time-base will mean that each measure will contain six data records, and the fifth data record will correspond to the onset of the third beat. Recall from Chapter 12 that the **yank -m** command allows us to extract particular data records following a specified marker. In the following command, we have defined the marker as a barline (`-m ^=`) and instructed **yank** to fetch the fifth line following each occurrence of the marker (`-r 5`). In our example, the **grep** command is being used to count V/V chords occurring on third beats:

```
timebase -t 8 strauss | solfa | yank -m = -r 5 | grep re \
   | grep fe | grep -c la
```

We can repeat this command for beats one and two by changing the **-r** parameter to 1 and 3 respectively.

Reprise

In this chapter we have learned how to concatenate musical passages together using the **cat** command. We also learned how to eliminate redundant exclusive and tandem interpretations from concatenated outputs using the **-u** and **-t** options for **rid.** In addition, we learned how to assemble two or more spines into a single output file using **assemble**. In the case of `**kern` and `**recip` representations, we learned how to use the **timebase** command to preprocess each constituent file so that all data records represent equivalent elapsed durations. Having assembled a full score from parts, **rid -d** can be used to eliminate any residual or unnecessary null data records. The **proof** command can be used to ensure that any assembled `**kern` data is correctly aligned.

Finally, we learned that the **timebase** command can be used for other analytic purposes. Specifically, it can be used to reduce a score rhythmically so only particular onset points or beats are retained. In Chapter 23 we will see additional uses of **timebase** for a variety of types of rhythmic tasks.