Chapter 20

Strophes, Verses and Repeats

We often tend to think of musical information as a linear stream of successive events. However, there are many circumstances where musical information exhibits more complex structures. These include such structural devices as repeats, da capos, first and second ending, multiple verses, alternative or *ossia* passages, different performance renditions, divergent sources, and competing editions or versions.

This chapter describes the basic Humdrum mechanisms for representing non-linear musical structures. The two critical mechanisms are the Humdrum *section* and *strophe*. We will encounter examples using the **yank**, **thru**, and **strophe** commands.

Section Labels

Musical scores are often notated to take advantage of repetitions in the music. Devices such as repeat marks, *Da Capo*, *Dal Segno*, *Codas*, and other mechanisms make it possible to represent a musical work in an abbreviated format. Humdrum provides corresponding 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
*>Exposition>2nd Theme
```

Notice that spaces can appear in section labels — as in `1st Ending`. Sections begin with a section label and generally end when another section label is encountered. Sections also end whenever 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.

Expansion Lists

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. The list is specified in square brackets. Like section labels, expansion lists begin with an asterisk followed by a greater-than sign. 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.

Using *yank* to Extract Sections

We encountered the **yank** command earlier in Chapter 12. Recall that **yank** can be used to extract material by *section* using the **-s** option. For example, if the appropriate section is labelled, we might extract the coda of a work as follows:

```
yank -s Coda -r 1 file
```

Recall that the **-r** option is manditory with **yank**; in this case, it identifies the *first* occurrence of a section labelled `Coda`.

Using the *thru* Command to Expand Encodings

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 result in further changes to the file.

Alternative 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 *ver-*

sion label. Consider the following expansion lists:

```
*>Gould1982[A,A,B]
*>Landowska[A,A,B,B]
```

Here we see two expansion lists, one carries the version label `Gould1982` and the other is labelled version `Landowska`. These expansion lists might encode different interpretations of the repeats in a rounded binary form — Landowska performed the second repeat whereas Gould (1982) did not. When the **thru** command is invoked, the user can specify which *version* is intended using the **-v** option. The appropriate through-composed expansion will be output.

The following example illustrates the use of the **thru** command in selecting particular versions of data in a file. Three sections are encoded in the file — labelled A, B and C. Each section in this example contains just a single data record. Three expansion lists are encoded: one is unlabelled, a second is labelled `long` and a third is labelled `weird`.

```
**example          **example
*>[A,B,A,C]        *>[A,B,A,C]
*>long[A,A,B,A,C]  *>long[A,A,B,A,C]
*>weird[C,A,C]     *>weird[C,A,C]
*>A                *>A
data-A             data-A
*>B                *>B
data-B             data-B
*>C                *>C
data-C             data-C
*-                 *-
```

Consider the following command:

```
thru -v weird file
```

The corresponding "through-composed" output would be as follows:

```
**example   **example
*thru       *thru
*>C         *>C
data-C      data-C
*>A         *>A
data-A      data-A
*>C         *>C
data-C      data-C
*-          *-
```

Notice that all expansion-list records have been eliminated from the output. A `*thru` tandem interpretation has been added to all output spines immediately following the exclusive interpretation. Also notice that there are now two sections in the output sharing the same label (`*>C`). This duplication of section-labels is not permitted in abbreviated-format encodings and can only occur in through-composed documents.

Without the **-v** option, **thru** expands the abbreviated file according to the *unlabelled* (default) ex-

pansion list. So the following command would result in an output consisting of section A, followed by section B, followed by section A (again), followed by section C:

```
thru file
```

Section Types

Suppose we had two different theorists — Smith and Jones — who had analyzed the same work differently. Smith thinks there are basically two sections in the work, whereas Jones argues that there are essentially three sections. Humdrum permits alternative schemes of section labels to coexist in a file by allowing the user to designate section *types*. A section label is considered to have a "type" when more than one greater-than sign (>) is present in the label. Consider the following example of sections defined by Smith and Jones:

```
**Example
*>Smith>A
*>Jones>A
data1
*>Jones>B
data2
*>Smith>B
data3
*>Jones>C
data4
*-
```

Both Smith and Jones label the work as beginning with section 'A'. Later Jones's 'B' section begins; then Smith's 'B' section; then Jones's 'C' section. Note that Smith's 'B' section also contains the material Jones has identified as section 'C'.

Normally, the **yank** command extracts a labelled section up to the next occurrence of a section label. However, the **-t** option causes **yank** to ignore all section labels except for a specified type. We could extract Smith's 'B' section by using the **-t** option to limit extraction to "Smith"-type section labels:

```
yank -t Smith -s B
```

This command would produce the following output:

```
**Example
*>Smith>B
data3
*>Jones>C
data4
*-
```

Without the **-t** option, **yank** will simply extract material up to the occurrence of the next section label. Note that section types can be used to define innumerable alternative organizations for a single document.

Hierarchical Sections

For many applications, it is useful to define "nested" structures where two or more sections form part of a larger section. Humdrum section labels allow users to distinguish hierarchical *levels*. Levels are indicated by the number of greater-than signs following the section type. Consider the following:

```
**Example
*>Form>Exposition
data1
*>Form>>1st Theme
data2
*>Form>>2nd Theme
data3
*>Form>Development
data4
*>Form>Recapitulation
*>Form>>1st Theme
data5
*>Form>>2nd Theme
data6
*>Form>Coda
data7
*-
```

All of the above section labels are identified as type `Form`. However, two levels are distinguished (denoted by > and >>). Subsections are specified by increasing the number of greater-than signs, hence `2nd Theme` is a subsection. When **yank** is invoked, it will extract the identified section up to the next section of comparable level. The operation is illustrated in the following sample commands: indicating the first and second themes.

```
yank -t Form -s '1st Theme' -r 1     (extracts up to >Form>>2nd Theme)
yank -t Form -s '2nd Theme' -r 1     (extracts up to >Form>Development)
yank -t Form -s 'Exposition' -r 1    (extracts up to >Form>Development)
```

For example, the second theme from the recapitulation can be extracted as follows:

```
yank -t Form -s '2nd Theme' -r 2
```

Alternatively:

```
yank -t Form -s Recapitulation file | yank -t Form -s '2nd Theme' -r 1
```

Using the *yank* and *thru* Commands

Section labels can be used in a wide number of applications. By way of illustration, here are a few pipeline processes involving section labels. First, we might ask the question — how does the user know what sections labels are present in a document? This is a task for **grep**:

```
grep '^\*>' file
```

This command will also output any expansion-lists. If we want to restrict our output to identifying which *versions* are available for a document we would look for the presence of square brackets:

```
grep '^\*>.*\[.*\]' file
```

How many notes are there in the exposition?

```
yank -t Form -s Exposition -r 1 file | census
```

How many phrases are there in the development?

```
yank -t Form -s Development -r 1 file | grep -c '{'
```

Extract the figured bass for the third recitative:

```
yank -s Recitativo -r 3 file | extract -i '**B-num'
```

Compare the estimated key for the second theme in the exposition versus the estimated key for the second theme in the recapitulation:

```
yank -t Form -s '2nd Theme' -r 1 file | key
yank -t Form -s '2nd Theme' -r 2 file | key
```

Determine the nominal (non-rubato) duration of Gould's performance of the work:

```
thru -v Gould1982 file | extract -i '**kern' | extract -f 1 \
    | dur -d | rid -GLId | grep -v '^=' | stats | grep -i total
```

Perform the first three measures from the second section of a binary form:

```
yank -s B file | yank -o = -r 1-3 | midi | perform
```

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) 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 exclusive interpretation. 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          **silbe
*k[b-e-]        *Deutsch
*               *solo
*>[1,1,1,1]     *>[1,1,1,1]
*>1             *>1
*               *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 section 1 is to be repeated 4 times in total.

The strophic passage pertains only to the spine marked `**silbe`. The `**silbe` representation pertains to syllabic text encoding and is a pre-defined representation in Humdrum. The `**silbe`

representation is discussed in Chapter 27. 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 *strophe* Command

The Humdrum **strophe** command can be used to isolate or extract selective strophic data. The **-x** option for **strophe** allows the user to extract a particular labelled strophe. Consider, for example the effect of the following command:

```
strophe -x 3 schubert
```

Using the above data, the result is:

```
!! Franz Schubert, 'Das Wandern' from "Die Schoene Muellerin"
**kern          **silbe
*k[b-e-]        *Deutsch
*>[1,1,1,1]     *>[1,1,1,1]
*               *solo
*>1             *>1
8f              Das
=5              =5
8f              sehn
8b-             wir
8a              auch
8ee-            den
=6              =6
(16dd           Rä-
16ff)           |
(16dd           -dern
16b-)           |
8f              ab,
8dd             den
=7              =7
(8.cc           Rä-
16a)            |
8b-             dern!
8r              %
*-              *-
```

Notice that all of the tandem interpretations related to the strophe organization are eliminated from the output.

Suppose that we wanted to create a through-composed version of the entire work. We would expect as output, just two spines — the `**kern` spine and the `**silbe` spine. First, we need to create the full length version using the **thru** command. This will take the default expansion list, and

repeat the appropriate section for each successive verse.

```
thru schubert
```

The effect of this will be to simply repeat section 1 four times. However, each repetition will contain all four verses. We can use the **strophe** command to eliminate the unwanted verse texts at each verse. When no option is given, **strophe** operates by preserving strophes in numerical order. That is, when it encounters the first strophic section it will preserve strophe #1 (`*S/1`); then when it encounters the next strophic section it will preserve strophe #2 (`*S/2`). And so on. In summary, the follow command will create a proper through-composed rendition of the Schubert lieder illustrated above.

```
thru schubert | strophe
```

Incidentally, the input passage need not necessary begin with strophe #1. The **strophe** command will adapt to the input, and use the lowest previously unencountered strophe number.

Using the *strophe* and *thru* Commands

As noted, the strophe technique can be used to encode different editorial interpretations of a single work. Suppose for example that we had two editions of the Bach chorale harmonizations: Erk and Reimenschneider. We could select the Erk edition as follows:

```
strophe -x Erk chorale166
```

In a strophic song, suppose we would like to compare the number of syllables in the first and second verses. We begin by selecting the appropriate verse, extract the syllable spine, eliminate all non-data records, eliminate any other special signifiers (like barlines), and finally count the number of remaining records. We repeat this procedure for both verses:

```
strophe -x 1 file | extract -i '**silbe' | rid -GLId \
    | grep -v [=\|%] | wc -l
strophe -x 2 file | extract -i '**silbe' | rid -GLId \
    | grep -v [=\|%] | wc -l
```

(In the `**silbe` representation, the vertical bar (|) and the percent sign (%) have special meanings so the **grep -v** is used to eliminate them along with barlines.)

Reprise

Between stophes and sections, highly non-linear musical documents can be constructed. We have seen how section labels can be defined, how lists of sections ("expansion lists") can be constructed and expanded to through-composed formats using the **thru** command. An unlabelled expansion list is the default version. Other versions have labelled expansion lists.

Several different *types* of section labels can coexist in the same document and the **yank** command can be instructed to ignore all sections other than a certain type via the **-t** option.

The basic ideas introduced in this chapter are summarized in the following table.

section	passage defined by a section label, ends with occurrence of section label of identical level
section label	tandem interpretation beginning: *> and not containing square brackets
section type	first part of section label: *>*type*>
expansion list	tandem interpretation beginning *> and containing a list of section labels in square brackets, e.g. *>[A,B,A]
version	a labelled expansion list, e.g. *>ternary[A,B,A]
level	hierarchical level of a section, designed by the number of '>' following the section type, e.g. *>type>>>name is lower than *>type>name
abbreviated format	Humdrum document encoded using expansion lists
through-composed	Humdrum document encoded without expanion lists
thru	command to create a through-composed document from an abbrevatiated format
thru -v	command to create a particular version of a through-composed document
yank -s	command to extract sections
yank -t -s	command to extract sections limited to sections of a particular type
strophe	1. alternative spine path, 2. command for extracting a particular strophe
strophic passage initiator	tandem interpretation indicating the beginning of a strophe (*strophe)
strophic passage terminator	tandem interpretation indicating the end of a strophe (*S-)
strophe label	tandem interpretation labelling one of several alternative spine-paths, begins *S/
strophe end indicator	tandem interpretation indicating the end of some spine path, e.g. *S/fin

Summary of terms related to sections and strophes

In Chapter 37 we will see further examples of how sections and strophes are especially useful when producing electronic editions.