10. Linking any number of databases together

The `relation` data type can represent references between any number of databases. In the first instance, you can set up references in one of two ways:

- On the one hand, you may already have *associated* two databases; i.e. they may contain identification numbers which refer to one another. In this case you enter the identification numbers as `text` information items while you are setting up the separate databases, and then use the `bridge` command to convert them into data of the `relation` data type, each referring to the other.
- On the other hand you can also link data together on the basis of other considerations, for example, based on the analysis of the relationships between two databases described in the previous section. You use individual `bridge` directives to define these point-to-point connections. Each directive creates a link between two points in the two databases.

10.1 Linking databases together: the `bridge` definition

Let us assume we have two sources, both of which have been prepared as $\kappa\lambda\epsilon\iota\omega$ databases. The first of them contains details of a series of people, including their taxpayers' reference numbers:

```
database name=source;first=person;overwrite=yes
part name=person;position=id,name,first_name,occupation,tax_ref_no
exit name=source
read name=source
person$document-1/Maier/Friedrich/tailor/a-123.c
person$document-2/Schrusser/Hans/doctor/a-146.d
person$document-3/Wilkog/Peter/farmer/a-234.a
person$document-4/Russen/Anna/widow/a-242.c
person$document-5/Bohlen/Paul/blacksmith/a-278.b
person$document-6/Ahrens/Johann/farmer/a-321.c
```

The second source, a tax list, shows the amount of tax paid for every taxpayer's reference number on the list, thus:

```
database name=tax_list;first=b;overwrite=yes
part name=taxpayer;position=id,tax_payment
exit name=tax_list
read name=tax_list
b$a-123.c/1fl 4x
b$a-146.d/5fl
b$a-234.a/2fl 12x
b$a-242.c/8x
b$a-278.b/3fl 20x
b$a-321.c/2fl 14x
```

In this example, one source contains items of information which are included in the second source as document identifications. From a formal point of view:

- the groups to be linked in one source are "absolutely addressable". These groups either open each document in question, or else they were introduced by a `part` directive in which you specified `identification=yes`.
- the other source contains elements of the `text` data type, which contain the identifications of the groups in the first source.

10.1.1 The `bridge` command

In this situation, you can use a `bridge` command to link the two databases.

This has the general form:

FIRSt=<filename>	**(MUST be specified!)**
SECOnd=<filename>	**(MUST be specified!)**
PART=<list of paths>	**(default: none)**
TYPE=Definitions | Confirmations	**(default: none)**
RELAtion=<name of definition>	**(default: reciprocal filename)**
BRIDge=<name of element>	**(default: reciprocal filename)**

Note: you should only define one of the two parameters part= or type=!

10.1.1.1 Links between associated databases

The following `bridge` command is required in order to link the two sources in our example:

```
bridge first=source;part=:tax_ref_no;second=tax_list
```

The three parameters used expect the following values:

- The `first=` parameter expects as a value the name of the database containing details of the link between the two databases in the form of the contents of one or more elements.
- The `part=` parameter expects as a value a list of paths with the names of all the elements which this `bridge` command should use in order to set up a link. The data element in question should refer to the database specified in `first=`.
- The `second=` parameter expects as a value the name of the database containing the character strings held in the entries defined by `part=` as identifications.

10.1.1.1.1 Properties of a standard link

This command is used to set up a linkage between the two databases in the following way:

An element named `tax_list` is introduced in the database named `source`. Ideally, this element's data type is `relation`, which refers to the `relation` definition `tax_list`. If you use this element in an evaluation command, the system assumes that it should be processed as a `relation` data item.

The above-mentioned `relation` definition is placed in the `source` database: an annotation is made to the effect that it is setting up a linkage to the `tax_list` database.

In every group in the `source` database containing an entry which can be linked to an identifier in the `tax_list` database, an entry is made for the `tax_list` element referring to the group in the `tax_list` database to whose identifier the group in the `source` database can be linked.

Exactly the same data is entered in the `tax_list` database in each case, but in such a way that it sets up a linkage to the `source` database. So once the `bridge` command has been executed, an element with the name `source` and the preferred data type `relation` – relating to the `source relation` definition – is also known in the `tax_list` database. In every group whose identifier can be linked to an entry in `tax_list`, an entry is made in the `tax_list` element which refers to the corresponding group in this database.

If you assign the name of one and the same database to both parameters, `first=` and `second=`, a linkage is constructed within this database which does not differ in any way from the linkages of the `relation` data type defined while the data was being input.

10.1.1.1.2 Modifying the standard linkage

The names of `relation` definitions and elements generated in this way can be changed by defining the appropriate `relation=` and `bridge=` parameters. Both parameters can only be defined after the first appearance of the `first=` or `second=` parameter. In this case, they both relate to the one of these two parameters which was last encountered. These may not be specifed more than twice.

- `relation=` is used to specify the name of the `relation` definition in the last database to be addressed.
- `bridge=` is used to define the name of the element in the last database to be addressed. Hence the `bridge` command

```
bridge first=source;part=:tax_ref_num;bridge=linkage;relation=tax;
          second=tax_list;bridge=person;relation=connections
```

  would have the following result:
- A new `linkage` element is incorporated into the `source` database which has the `relation` data type as its preferred data type and uses the `relation` definition `connections`.

10.1.1.1.3 Exceptions

You can use the two parameters we have just explained in order to set up the linkage between two databases using names which are easier to remember or interpret. Otherwise they are only important if you want to link databases with names identical to those of the elements or `relation` definitions contained in them at the present moment.

In this case (if you are not using `bridge=` or `relation=`), the following applies:

If a `bridge` command is used to introduce an existing element, a note is added to this element, drawing attention to the fact that entries of the `relation` data type are related to the `relation` definitions with the corresponding name. *However, these elements still contain the data type which has been preferred up to now. If you want to use them as* `relation` *data, you should therefore use the* `relation[]` *function if appropriate.*

If an existing `relation` linkage is addressed by the linkage between the two databases, it is supplemented in such a way that elements referring to it can also refer to the newly linked database.

10.1.1.2 Linking databases which have not been associated

Linkages between databases can also be made regardless of the information items contained in them. Let us assume that the first source we used as an example is to be linked to another source, e.g. an address list:

```
database name=addresses;first=person;overwrite=yes
part name=person;position=ser.no.,name,firstname,address
exit name=addresses
read name=addresses
person$1/Ahrens/Johann/Kirchengasse 27
person$2/Bohlen/Paul/Hauptplatz 12
person$3/Maier/Friedrich/Hauersteig 14
person$4/Russen/Anna/Kirchengasse 18
person$5/Schrusser/Hans/Am Freithof 2
person$6/Wilkog/Peter/Vor dem Berge 39
```

In this case, the `part=` parameter in the `bridge` command must first be replaced by the `type=` parameter with a value of `definitions`. This indicates that the linkages between the two databases should be made by means of explicit definitions, for example, thus:

```
bridge first=source;second=addresses;type=definitions
```

Written in this way, the `bridge` command opens a definition with the same name; the command is followed by `bridge` directives which each define one linkage between two groups in the databases concerned and are closed by an `exit` directive containing the keyword `definitions` in its `name=` parameter. So we can set up six linkages, written something like this:

```
bridge first=source;second=addresses;type=definitions
bridge
bridge
bridge
bridge
bridge
bridge
exit name=definitions
```

10.1.1.2.1 The `bridge` directive

These directives have the general form:

FIRSt=<symbolic address>	**(MUST be specified!)**
SECOnd=<symbolic address>	**(MUST be specified!)**
ONLY=<number>	**(default: none)**
VISIbility=<number>	**(default: none)**

10.1.1.2.1.1 Basic format of a `bridge` directive

In most cases, however, only two of these four parameters are ever used. `first=` and `second=` both include a constant – i.e. between quotation marks or inverted commas – consisting of a definition stating where in the particular database the linkage should be set up. In its simplest form, a "symbolic address" of this kind consists of a document identifier without any additional information. So, continuing with our example, in order to create a linkage between `document-1` in the `source` database (containing basic information about Mr. Friedrich Maier) and his address in document `3` in the `addresses` database, we require the following `bridge` directive:

```
bridge first="document-1";second="3"
```

From this, we obtain the following `bridge` definition to connect our two small databases:

```
bridge first=source;second=addresses;type=definitions
bridge first="document-1";second="3"
bridge first="document-2";second="5"
bridge first="document-3";second="6"
bridge first="document-4";second="4"
bridge first="document-5";second="2"
bridge first="document-6";second="1"
exit name=definitions
```

If you write the `bridge` command, opening up the `bridge` definition, in this way, it makes no practical difference whether you refer to a database in `first=` or `second=`. It is important, however, that all the documents quoted in the `bridge` *directives* following the `first=` parameter really do relate to the database quoted in `first=` in the `bridge` *command*; the same applies to the `second=` parameter of the command or of the directives.

10.1.1.2.1.2 Symbolic addresses

Symbolic addresses are a platform-independent means of defining *where* a particular group is in a database. Since you can use a document's identification to give access to the group initialising this document, the document identifier is at the same time the document's symbolic address and the simplest kind of symbolic address there is. We have already given some examples of how it can be used.

However, because you can only use this particular method to access a document as a whole, the element representing such a symbolic address is also always present in the group initialising the document. For this reason, as a general rule you should always specify addresses more precisely in structured databases.

In the following examples, we will work with the database fragment given below:

```
database name=census;
        first=house
part name=house;
        position=housenumber;
        part=flat
part name=flat;
        position=flatnumber;
        part=hdofhousehold,spouse,son,daughter
part name=hdofhousehold,spouse,son,daughter;
        position=familyname,firstname
element name=flatnumber;identification=yes
exit name=census
read name=census
house$123a
flat$a
hdofhousehold$Müller/Friedrich
spouse$Müller/Johanna
son$Müller/Fritz
son$Müller/Josef
daughter$Müller/Maria
daughter$Müller/Anna
flat$b
hdofhousehold$Meier/Andreas
spouse$Meier/Katherina
son$Meier/Wilhelm
son$Meier/Hans
daughter$Meier/Ulrike
```

In this database, the following `bridge` directive

```
bridge first="123a"
```

obviously refers to the house as a whole. Every resulting element becomes a constituent of the group `house`.

Because all groups invoked in the rest of the sequence belong to this house, all their symbolic addresses also begin with the character string we have just quoted. If you want to distinguish between the flats in the house, you must append the identifier of the group "flat", which depends directly on the document, to the document identifier after an slash.

```
bridge first="123a/flat"
```

If you do not add any additional specifications, this always refers to the group's first appearance in the document. In our example then, it refers to flat `a`, the flat owned by the Müller family.

There are two ways we can refer to flat `b`, owned by the Meier family.

On the one hand, we can specify the ordinal number of the requisite flat, after a semi-colon:

```
bridge first="123a/flat;2"
```

On the other hand, we can specify the group identifier of the requisite flat after an equals sign. Because the element `flatnumber` was defined as the group identifier for this group, we can write the following:

```
bridge first="123a/flat=b"
```

At this point, note that the two following statements agree with what we have just been saying:

```
note The flat owned by the Müller family can also be described
note in detail by means of the expression
bridge first="123a/flat=a"
note or
bridge first="123a/flat;1"
```

You can continue to apply the principle we have just described to as many levels in the hierarchy as you like. Please satisfy yourself that the following symbolic addresses match the principles described above:

```
note The symbolic address for Friedrich Müller goes as follows:
bridge first="123a/flat/hdofhousehold"
note The symbolic address for Hans Meier is as follows:
bridge first="123a/flat=b/son;2"
note The symbolic address for Ulrike Meier goes as follows:
bridge first="123a/flat;2/daughter"
note The symbolic address for Josef Müller is as follows:
bridge first="123a/flat;1/son=son-2"
```

10.1.1.2.1.3 Weighted `bridge` directives

The two `bridge` directive parameters we have not yet discussed can be used to assign a specific visibility to a linkage you have defined, or assign a particular view to it.

- The `visibility=` parameter expects a value consisting of
  - a question mark, which is assigned a value of 0.5, or
  - an exclamation mark, which is assigned a value of 0.95, accompanied by
  - a number between zero and one, which is accepted as it is, or
  - a number greater than one, which is repeatedly divided by one hundred until the result lies between zero and one.

This parameter specifies that the visibility of the linkage between the two databases being processed corresponds to the given numerical value.

- The `only=` parameter expects a value consisting of a whole number between one and the largest alternative level code valid on the platform in question. (Typical values of this kind include 8, 16, 32 etc.)

It specifies that the linkage between the two databases being processed only exists in the view with the given level code.

10.1.1.2.2 Once you have set up linkages ...

Linkages set up by a `bridge` command with the `type=definitions` exactly correspond to those set up by a `bridge` command with the `part=` parameter. All additional parameters and exceptions explained there also apply in this instance.

10.1.1.3 Confirming proposed linkages

The parameter value `conformations` in the `bridge` command's `type=` parameter will give you the option of linking groups from different databases – groups which the computer has suggested are potentially identical – together. This feature has not yet been released.

10.2 `relation` definitions referring to other databases

In particular cases (above all when linking a large number of databases together), it may be desirable for purposes of organisation to define which databases a given `relation` definition can link to the database for which this definition is being made – before you actually link the databases together.

For this purpose, the user can use the `bridge` directive in `relation` definitions.

It accepts the `name=` parameter as its only parameter. The latter expects the name of a database as a value, to which the database to which this `relation` definition belongs is supposed to be linked in the course of time. The database referred to does not have to exist at this moment in time.

You can use this directive as often as you like in a `relation` definition. However each directive can only be used to prepare a linkage with a single database.

So in order to prepare the structural organisation whereby three sources are linked together you can use, for example, the following `relation` definition:

```
item name=linkages;usage=relation;source=example;type=permanent
bridge name=tax_list
bridge name=addresses
exit name=linkages
```

10.3 Managing linked databases

If you have linked several databases together, it is generally advisable to ensure that they are simultaneously available on magnetic disk. $\kappa\lambda\epsilon\iota\omega$ sometimes addresses a database which is linked to other databases even when the user has not explicitly requested it to. This happens:

1) If when processing one of the two elementary functions `each[]` or `total[]` in the context of a `write` command, the system encounters an entry which refers to another database. If the relevant database is not available, an appropriate message is displayed and the system continues to process the original database.
2) If you want to delete a database which is linked to other databases either explicitly – by issuing a `delete` command or implicitly by including `overwrite=yes` in a `database` command - $\kappa\lambda\epsilon\iota\omega$ attempts to remove all references to this databases from the databases which were linked to it. If the system is unable to address one of these databases at this point in time it does not delete the original database, but all references to it in all databases *accessible* to the system are removed. The missing database should then be made accessible and the system should then attempt to delete the one from which it started. *If in this situation you delete the "awkward" database using the operating system's own resources, and then recreate it at a later date, you may find that serious inconsistencies in the processing of linked databases result.*