bib2gls: sorting

Nicola L. C. Talbot

Abstract

When using makeindex and xindy, it's advisable to provide a sort value when the actual value contains commands or other awkward content that can confuse sorting. With bib2gls, which was written specifically for the glossaries-extra package, the advice is the reverse. In general you shouldn't explicitly supply the sort value but instead make use of bib2gls's system of fallbacks. This provides a more flexible approach that makes it easier to share bib files across multiple documents that may require different ordering.

1 Sorting with \makeglossaries

Symbols can be quite difficult to order. Consider the following document that just uses the base glossaries package [9]:

```
\documentclass{article}
\usepackage[style=treegroup]{glossaries}
\makeglossaries
\loadglsentries{constants}
\begin{document}
\gls{pi}, \gls{e}, \gls{gelfondcons} and
\gls{root2}.
\printglossary[nonumberlist]
\end{document}
```

The file constants.tex contains the following:

```
\newglossaryentry{pi}{name={\ensuremath{\pi}},
description={ratio of circumference of a circle
 to its diameter},symbol={3.14159}}

\newglossaryentry{e}{name={\ensuremath{e}},
description={Euler's number},symbol={2.71828}}

\newglossaryentry{root2}{
name={\ensuremath{\surd 2}},symbol={1.41421},
description={Pythagoras' constant}}

\newglossaryentry{gelfondcons}{
name={\ensuremath{e\sp\pi}},symbol={23.1406926},
description={Gelfond's constant}}
```

I've used the symbol key to store the approximate value, which means that it can be shown in parentheses in the glossary with the treegroup style.

The sort key hasn't been explicitly set, so its value is obtained from the name; but further, since makeindex doesn't recognise (LA)TEX commands, it treats \ensuremath as a literal backslash followed by 10 letters. The initial backslash results in the entry being placed in the 'symbols' group. All four sort values start with \ensuremath{ so the relative ordering of those terms is based on the 13th character onwards (backslash comes before 'e' but after '}'): \p, \s, e} and e\ (Figure 1).

Glossary

Symbols

π (3.14159) ratio of circumference of a circle to its diameter.
$\sqrt{2}$ (1.41421) Pythagoras' constant.
e^{π} (23.1406926) Gelfond's constant.
e (2.71828) Euler's number.

Figure 1: Default ordering with makeindex

Glossary

Numbers

$\sqrt{2}$ (1.41421) Pythagoras' constant.

E

e (2.71828) Euler's number.

P

π (3.14159) ratio of circumference of a circle to its diameter.

Figure 2: Ordering with xindy (π has sort=pi and Gelfond's constant uses \sp)

If I switch to xindy (which requires adding the xindy package option) then the document build will fail: xindy discards commands and characters such as { } and $, which means that the sort value for the pi entry ends up as an empty string, which xindy doesn't allow. A sort value that's acceptable to xindy must be provided. For example:

```
\newglossaryentry{pi}{name={\ensuremath{\pi}},
sort={pi},
description={ratio of circumference of a circle
 to its diameter},symbol={3.14159}}
```

This will place the π entry in the 'P' letter group (Figure 2).

The $\sqrt{2}$ entry ends up in the 'numbers' group because once the commands and braces have been stripped only '2' is left. Similarly, 'e' is all that remains for both Euler's number and Gelfond's constant. Since xindy merges items with duplicate sort values this means that the locations from Gelfond's constant ends up merged into Euler's number location list. In this simple example, the locations are all page 1 and they've been suppressed with the nonumberlist option so it appears as though Gelfond's constant has been ignored.

If I use ^ instead of \sp then the sort value for Gelfond's constant becomes e^ instead of just e, which now means all sort values are unique (Figure 3).

doi.org/10.47397/tb/42-2/tb131talbot-sorting

Glossary

Numbers

$\sqrt{2}$ (1.41421) Pythagoras' constant.

E

e (2.71828) Euler's number.
e^{π} (23.1406926) Gelfond's constant.

P

π (3.14159) ratio of circumference of a circle to its diameter.

Figure 3: Ordering with `xindy` (π has sort=pi and Gelfond's constant uses `^`)

Let's suppose now that there's a chance that I might have an editor who insists on using an upright font for constants. Providing some commands will make it easier to switch:

```
\newcommand{\constante}{\mathrm{e}}
\newcommand{\constantpi}{\uppi}
```

(`\uppi` requires upgreek [3].) Now `e` and `\pi` need to be replaced with `\constante` and `\constantpi` in the `name` values. This means that with `xindy` the `e` entry will end up with an empty sort value. This will also happen to Gelfond's constant if `\sp` is used. If `^` is used instead then this will end up as the only character in the sort value.

This means that, particularly with `xindy`, entries that are symbols will typically need to have the `sort` key set as they are likely to degrade into empty strings or non-unique values. In the case of these constants, their approximate numeric values may be a more appropriate sort value. A helper command can make it easier to assign. For example:

```
\newcommand{\newconstant}[5][]{%
 \newglossaryentry{#2}{name={#3},symbol={#4},
description={#5},sort={#4},#1}}
\newconstant{pi}{\ensuremath{\constantpi}}
 {3.14159}{ratio of circumference of a
  circle to its diameter}
\newconstant{e}{\ensuremath{\constante}}
 {2.71828}{Euler's number}
\newconstant{root2}{\ensuremath{\surd 2}}
 {1.41421}{Pythagoras' constant}
\newconstant{gelfondcons}
 {\ensuremath{\constante^\constantpi}}
 {23.1406926}{Gelfond's constant}
```

Although `makeindex` can numerically order integers, it doesn't recognise decimals, so all the entries end up in the 'symbols' group ordered according to a string comparison, where '23' comes between '2.' and '3.' (Figure 4).

Glossary

Symbols

$\sqrt{2}$ (1.41421) Pythagoras' constant.
e (2.71828) Euler's number.
e^{π} (23.1406926) Gelfond's constant.
π (3.14159) ratio of circumference of a circle to its diameter.

Figure 4: Ordering with `makeindex` by approximate value

Glossary

Numbers

$\sqrt{2}$ (1.41421) Pythagoras' constant.
e^{π} (23.1406926) Gelfond's constant.
e (2.71828) Euler's number.
π (3.14159) ratio of circumference of a circle to its diameter.

Figure 5: Ordering with `xindy` by approximate value

With `xindy`, by default the entries end up in the 'numbers' group (since the sort values all start with a digit) and digits come before punctuation so '23' is placed between '1.' and '2.' (Figure 5). In order to sort numerically with `xindy`, it's necessary to use the `numeric-sort` module. You can either call `xindy` directly with `-M numeric-sort` or add the following to the document preamble:

```
\GlsAddXdyStyle{numeric-sort}
```

This now produces the desired result (Figure 6).

If I change my mind and decide to order by the description, I can simply change the definition of `\newconstant`. Other possibilities are to order by definition or by first use in the document (which require the sort=def or sort=use package options). These options work by assigning a numerical (integer) value to the `sort` key that corresponds to the desired order. The value is zero-padded in the event that `xindy` is called without the `numeric-sort` module.

Suppose I now want to switch to using `bib2gls` [7] with glossaries-extra [8]. Ordering by definition or use can now be indicated with the resource options (not package options) sort=unsrt or sort=use.

Glossary

Numbers

$\sqrt{2}$ (1.41421) Pythagoras' constant.
e (2.71828) Euler's number.
π (3.14159) ratio of circumference of a circle to its diameter.
e^{π} (23.1406926) Gelfond's constant.

Figure 6: Ordering with `xindy -M numeric-sort` by approximate value

No comparisons are required in these cases, as it's simply a matter of iterating over the list of entries obtained from parsing the `bib` file or the list of records obtained from parsing the `aux` file.

Since all entries now have to be defined in the `bib` file, it's not possible to define a command like `\newconstant`, but `bib2gls` provides a flexible way of determining what the sort value should be.

2 `bib2gls` fallbacks

`bib2gls` has a set of fallbacks that are used if it needs to access a field which hasn't been set. The different entry types have different fallbacks. For example, when sorting entries the default behaviour is to obtain the sort value from the `sort` field. If this field *has not been set* then the value is obtained from the `sort` field's fallback. In the case of `@entry`, the fallback is the value of the `name` field. In the case of `@symbol` and `@number`, the fallback is the entry label (as given in the `bib` file).

If the fallback is also missing, then the fallback's fallback is used (if one is available) and so on. For example, consider the entry defined as:

```
@index{duck}
```

This only has a label (`duck`) and no fields. So when `bib2gls` tries to access the `sort` field and finds that it hasn't been set, it then tries the fallback for the `sort` field, which is the value of the `name` field for this entry type. The `name` field also hasn't been set, so the fallback for that field is required, which is the entry label. Therefore the sort value ends up as 'duck'.

Now consider

```
@indexplural{duck}
```

Again the fallback for the missing `sort` field is the value of the `name` field, which is also missing, but now the fallback for the `name` field is the value of the `plural` field, which is also missing. The fallback for `plural` is the value of the `text` field with the letter 's' appended. The fallback value for the `text` field is the entry label. Therefore the sort value ends up as 'ducks'.

Now consider

```
@index{glossary,plural={glossaries}}
@entry{gloscol,
  parent={glossary},
  description={collection of glosses}
}
@entry{gloslist,
  parent={glossary},
  description={list of technical words}
}
```

The sort value for the `gloscol` entry is obtained as follows:

1. Look up the value of the `sort` field. This isn't set, so use the fallback value, which is the value of the `name` field.
2. The `name` field isn't set, so use the fallback value for that, which is the parent entry's name.
3. The `parent` field provides the parent's *label* (`glossary`), so look up the value of the `name` field for the parent entry.
4. The `name` field isn't set for the `glossary` entry, so use the fallback value for that, which is the entry's label.

Therefore the sort value ends up as 'glossary'. The same process for `gloslist` leads to the same sort value.

It's possible to change the default fallbacks, but some fields, such as `description`, don't have a fallback, so that will terminate a fallback trail.

If the `sort` field is explicitly set, then *the fallback is not required* so in that situation changing the system of fallbacks has no effect. The recommendation is that you don't explicitly set the `sort` field but instead use the fallback system to choose the most appropriate field according to the entry type.

If you select a different field for the sort value, then that field's fallback (if provided) will be used instead; e.g., with the option sort-field=description then any entries which don't have the `description` field set will have an empty sort value (since there's no fallback for this field).

3 Examples

The examples below all use the same set of `bib` files but use different settings to adjust the order. For brevity, all entries are selected with no locations. The condensed style is designed to show the ordering in as compact a form as possible, for illustrative purposes only. The document fonts are set with:

```
\usepackage[light,condensed,math]{iwona}
\usepackage[T1]{fontenc}
```

The name is formatted in a bold font, but this will not be visible for mathematical content or pictographs. If the `symbol` field is set, it's shown in parentheses before the description. (Bold parenthetical content is part of the entry's name.) The upgreek and marvosym [2] packages are required for some of the symbols.

The entry definition set up in the preamble for each example is:

```
\setabbreviationstyle{long-short-sm-desc}
\setabbreviationstyle[acronym]{nolong-short-em}

\GlsXtrLoadResources[
 selection=all,save-locations=false,
 ⟨options⟩]
```

This uses the 'sm' abbreviation style for entries defined using `@abbreviation`, which requires the relsize package [1]. The entries defined using `@acronym` will use the 'em' abbreviation style which formats the short form using `\emph`.

Additional `\GlsXtrLoadResources` commands may be present for some examples. The main body of the document just contains `\printunsrtglossary`. Sample entries from each `bib` file are shown below. The complete `bib` files can be downloaded [4].

`abbreviations.bib` contains entries such as:

```
@abbreviation{xml,
  short={XML},
  long={extensible markup language},
  description={a markup language that defines
        a set of rules for encoding documents}
}
@acronym{nasa,
  short={NASA},
  long = {National Aeronautics and Space
          Administration}
}
```

`constants.bib` contains entries such as:

```
@number{pi,
  description={pi},
  name={\ensuremath{\constantpi}},
  symbol={3.14159}
}
@number{root2,
  description={Pythagoras' constant},
  name={\ensuremath{\surd2}},
  symbol={1.41421}
}
@number{zero,
  description={zero},
  name={\ensuremath{0}}
}
```

As with the earlier `makeindex` and `xindy` examples, the `symbol` field has been used to store the approximate values so that they can easily been seen in the glossary.

The custom commands such as `\constantpi` are also provided:

```
@preamble{"
\providecommand{\constanti}{\mathrm{i}}
\providecommand{\constante}{\mathrm{e}}
\providecommand{\constantpi}{\uppi}
\providecommand{\constantgamma}{\upgamma}
\providecommand{\constantphi}{\upphi}
\providecommand{\constantlambda}{\uplambda}"}
```

These definitions can be detected by `bib2gls` and will be used if they are encountered within any sort values.

`entries.bib` contains entries such as:

```
@entry{mineral,
  name = {mineral},
  description = {solid, inorganic,
                 naturally-occurring substance}
}
@entry{quartz,
  parent = {mineral},
  name = {quartz},
  description = {hard mineral consisting
                 of silica}
}
```

`pictographs.bib` contains entries such as:

```
@symbol{heartsuit,
 name={\ensuremath{\heartsuit}},
 description={heart}
}
@symbol{phone,
  name={\Mobilefone},
  description={mobile phone}
}
```

`terms.bib` contains a mixture of `@index`, `@indexplural` and `@entry`, such as:

```
@index{sample}
@indexplural{homograph}
@entry{diamondjubilee,
  name={diamond jubilee},
  description={sixtieth anniversary}
}
```

It also uses an unknown entry type, for example:

```
@homograph{mineral.diamond,
  name={diamond},
  description={metastable allotrope of carbon}
}
@homograph{shape.diamond,
  name={diamond},
  description={four-sided shape with
               equal sides}
}
```

These entries will be ignored unless they are aliased to an entry type that `bib2gls` recognises.

3.1 Default sorting

The first example document uses the default sort settings, but it needs to alias the custom `@homograph` entries to make them behave as though they'd been defined with `@entry` instead:

```
src={terms,pictographs,abbreviations,constants},
entry-type-aliases={homograph=entry}
```

Since no sort method has been specified and there's no document language, the sort method will be alphabetical according to my locale (en-GB). The `--group` switch is used when invoking `bib2gls`. The result is shown in Figure 7.

Note that the entries defined with `@symbol` and `@number` have been ordered according to their label. (For example, `root2` and `phone`.) The homographs (such as 'diamond') trigger a warning from `bib2gls`:

```
Identical sort values for 'shape.diamond' and
'mineral.diamond'
Falling back on ID
```

This means that the *relative* ordering of the homographs is based on their labels (using a simple character code comparison) so the mineral diamond is placed before the shape diamond. Both will still have 'diamond' as the sort value when compared with other entries.

The action to perform in the event of duplicate sort values can be changed by setting a value for identical-sort-action. For example, you can order them according to first use in the document (which doesn't make sense for this example) or according to which entry was defined first in the `bib` file. You can also choose another field to determine the final relative ordering, but only a simple character code comparison is used (not a locale-sensitive alphabetical comparison).

If you have a set of homographs and you want to use a field containing natural language to determine their relative order then you may prefer to use the sort-suffix field instead, which can append the contents of another field to the sort value. This suffix will apply to all sort values, not just the homographs.

The abbreviations have been ordered according to the short form. This means that both XHTML and XML are placed in the 'X' letter group, even though the abbreviation style chosen in the document shows the long form first. This ordering is, however, appropriate for the acronyms such as 'Ofcom' and 'Ofsted'.

3.2 Sort suffix and fallbacks

The next example makes some adjustments to the resource options:

```
src={terms,pictographs,abbreviations,constants},
entry-type-aliases={homograph=entry}
sort-suffix=description,
symbol-sort-fallback=name,
abbreviation-sort-fallback=long
```

This will result in an error from inputenc arising from the upright Greek letter `\uppi`. The sort fallback for the symbol entries has been switched to the `name` field. Although the letter group *label* is numeric, `bib2gls` attempts to assign an appropriate title, which it obtains from the sort value of the first entry to be assigned to that group (in this case π).

Since the sort method is using my en-GB locale, the upright Greek letters will all be placed in their own group at the end because there's no rule for them in the English comparator being used. Since this final group is one that you will typically need to change, `bib2gls` provides a command to make it easy to do this:

```
\newcommand{\bibglssetlastgrouptitle}[2]{%
  \glsxtrsetgrouptitle{#1#2}{Greek}}
```

It's necessary to define this command *before* calling `\GlsXtrLoadResources`, or it will have no effect.

The result is shown in Figure 8. This has still produced some oddities.

As already mentioned, the Greek letters are all at the end of the glossary. The en-GB comparator recognises them as letters (rather than punctuation or other symbols) but they don't form part of the en-GB alphabet so they are all lumped together in a single group.

Some of the symbols have been placed in the 'symbols' group (0, 1, $\sqrt{2}$). This is because `bib2gls` recognised the commands in the `name` field (which is now being used as the sort fallback for symbols) for those entries and was able to convert them into Unicode. The comparator being used (en-GB) recognised those Unicode characters as symbols.

A search of `bib2gls`'s transcript file shows that `bib2gls` was also able to interpret the card suit commands but not the marvosym commands. For example, consider the `phone` entry:

1. The `sort` field hasn't been set, so use the fallback for `@symbol` (which is now the `name` field): `\Mobilefone`. Since `bib2gls` doesn't recognise this command the sort value is empty.
2. The sort-suffix=description setting then appends the contents of the `description` field, so the sort value is now `mobile phone`.

This means that the `phone` entry ends up in the 'M' letter group. Now consider the `heartsuit` entry:

1. The `sort` field hasn't been set so use the fallback for `@symbol` (which is now the `name` field): `\ensuremath{\heartsuit}`. These commands are recognised by `bib2gls` and are converted into the Unicode character U+2661. So the sort value consists of the single character ♡.
2. The sort-suffix=description setting then appends the contents of the `description` field, so the sort value is now '♡heart' but the en-GB comparator considers the heart character as ignorable punctuation and so the sort value becomes 'heart'.

This means that the `heart` entry ends up in the 'H' letter group.

The sort fallback value for abbreviations is given by the setting of abbreviation-sort-fallback. However,

this is used by both `@abbreviation` and `@acronym`. In this case, it's necessary to differentiate between `@abbreviation` (which needs to be sorted according to the long form) and `@acronym` (which needs to be sorted according to the short form). This requires using custom-sort-fallbacks instead, which can also be used to differentiate between `@number` and `@symbol`.

The sort-suffix option has also caused 'diamond jubilee' to be placed before the shape 'diamond'. This is because the description is now included in the sort value but by default no separator is inserted before the suffix. So the `shape.diamond` entry starts by fetching the sort fallback value from the `name` field ('diamond') and then appends the description so the sort value becomes 'diamondfour-sided shape with equal sides'. The default break-at=word setting marks the word boundaries so the final sort value is:

```
diamondfour-sided|shape|with|equal|sides|
```

The `diamondjubilee` entry starts by fetching the sort fallback value from the `name` field 'diamond jubilee' and then appends the description so the sort value becomes 'diamond jubileesixtieth anniversary'. Again the default break-at setting marks the word boundaries so the final sort value is:

```
diamond|jubileesixtieth|anniversary
```

The pipe character comes before the letter 'f' so 'diamond jubilee' ends up before 'diamond'.

This can be fixed either by switching to using identical-sort-action or by inserting a marker before the suffix. This marker would need to be a punctuation character or symbol that the comparator recognises as coming before letters. It also needs to come before the word boundary marker and should not be a character that's discarded by the collator.

3.3 Sort suffix marker and custom fallbacks

The next example needs to remove the definition of `\bibglssetlastgrouptitle` since the modified settings will now place the Greek characters in the 'symbols' group.

This example still uses sort-suffix to append the description to the sort value. I've chosen digits for the markers to ensure that they're not discarded by the alphabetical collator and word separator used by the sort method.

The resource options are now:

```
src={terms,pictographs,abbreviations,constants},
entry-type-aliases={homograph=entry}
sort-suffix=description,sort-suffix-marker=0,
break-marker=1,symbol-sort-fallback=name,
abbreviation-sort-fallback=long,
custom-sort-fallbacks={acronym=short,
                       number=symbol}
```

The result is shown in Figure 9. The entries defined with `@number` now use the `symbol` field to obtain the missing sort value. Unlike `makeindex` and `xindy`, the alphabetical sort methods don't create a 'numbers' group for items with numeric sort values but instead use the 'symbols' group for any entries that don't belong to a letter group.

The numbers aren't in numeric order (23.140692 is between 2.71828 and 3.14159). In fact, since the sort-suffix option has been set, the sort values for the constants aren't simple numbers but are the number followed by the description. The digit markers (0 and 1) have also caused the pictographs to appear between Euler's constant (0.57721) and Apéry's constant (1.2020569).

There are two entries defined with `@number` that don't have the symbol field set (one and zero). However, the sort-suffix setting appends the description so they end up ordered according to their description. (If you try this example and find them in the 'N' letter group, you need to update your version of `bib2gls`.)

3.4 Aliasing and concatenation

The sort-suffix option is turning out to be quite problematic for this example set of entries. The reason for using it was to ensure that the homographs were sorted by name and then description. The option identical-sort-action=description could be used as an alternative, but it won't allow for a locale-sensitive word sort of the description.

Each item in the custom-sort-fallbacks list has the general format:

⟨*original entry type*⟩=⟨*field1*⟩+⟨*field2*⟩...+⟨*fieldN*⟩

where ⟨*original entry type*⟩ is the entry type *as specified in the* `bib` *file*. The homographs were defined in the `bib` file using a custom entry type `@homograph`, which is then aliased to `@entry`. If I use `entry` within custom-sort-fallbacks it will only apply to entries that were defined within the `bib` file with `@entry` (not the entries that were aliased to `@entry`).

If I want to specifically change the sort fallback for entries defined with my custom `@homograph` without altering the fallback for the other entries then I need to use `homograph` for ⟨*original entry type*⟩, and I can use the concatenation operator (+) to create a fallback value that's formed by combining multiple fields. The default separator is a space but may be changed with field-concat-sep.

So this example dispenses with sort-suffix and relies instead on aliasing and the custom sort fallback. The resource options are now:

```
src={terms,pictographs,abbreviations,constants},
entry-type-aliases={homograph=entry},
```

```
symbol-sort-fallback=description,
field-concat-sep={.},
custom-sort-fallbacks={abbreviation=long+short,
number=symbol+name,homograph=name+description}
```

The result is shown in Figure 10. Note that I've had to change the default field concatenation separator; otherwise, the default space would result in 'diamond jubilee' (which now doesn't include the description in the sort value) being placed between the two 'diamond' entries (which do have their descriptions in the sort value).

The sort-symbol-fallback setting ensures that the pictographs are sorted according to their descriptions but this setting is overridden by custom-sort-fallbacks for the `@number` entries. Concatenating the `symbol` and `name` fields means that the 'zero' and 'one' entries (which don't have the `symbol` field set) are sorted according to their `name` fields and so are now in the 'symbols' group. However, the alphabetical word ordering means that they're not ordered numerically.

3.5 Sub-blocks

As discussed in the previous *TUGboat* article [5], `\printunsrtglossary` simply iterates over all defined entries for the given glossary. When used with `bib2gls` the entry definitions are in the `.glstex` file that's loaded by `\GlsXtrLoadResources`. If there's more than one instance of this command, each `.glstex` file is input sequentially and the entry labels are added to the internal list associated with the corresponding glossary.

As a result, a single `\GlsXtrLoadResources` command doesn't have to correspond to a single glossary. It may be used to process multiple glossaries at the same time (if there's some way of assigning the glossary type) or it may be used to process a sub-block of a single glossary. Each sub-block may be sorted according to a different method. The ordering of the sub-blocks corresponds to the order of `\GlsXtrLoadResources` commands.

When dividing the glossary into sub-blocks, it's possible for letter groups to become fragmented. For example, if my first sub-block contains ant, bee and zebra and the second sub-block contains aardvark, duck and wombat, there will be two 'A' letter groups. This is a contrived example, as it would result in a glossary with the rather odd order: ant, bee, zebra, aardvark, duck, wombat. It's more usual to use different sort methods for each sub-block, which may form different groups. Alternatively, you can override the sort method's group formation and force all entries in a sub-block to belong to a single group.

This next example will have multiple sub-blocks. The first block will be for the mathematical constants, ordered by the numerical value. The approximate value can be obtained from the `symbol` but, as noted above, this doesn't deal with 0 and 1. There are two ways of approaching this:

- the fallback can be made from a combination of the symbol and name (as in the previous example) and then strip any non-numeric content;
- if the `symbol` hasn't been set then copy the name into it.

The first method can be achieved with the resource options:

```
symbol-sort-fallback=symbol+name,
sort-replace={{[^0-9\string\.\string\-]+}{}}
```

The second method can be achieved with the resource options:

```
symbol-sort-fallback=symbol,
replicate-fields={name=symbol}
```

The only problematic entry is $\sqrt{-1}$, which is a complex number and therefore doesn't have a defined order within a set of real numbers. Whilst the TEX parser library used by `bib2gls` recognises `\surd` it doesn't recognise `\sqrt`. Unrecognised commands are ignored, so the sort value ends up as `-1`.

Here, I use the first method to avoid altering the `symbol` field. The resource command is:

```
\GlsXtrLoadResources[
 selection=all,save-locations=false,
 src={constants},sort=double,
 symbol-sort-fallback=symbol+name,
 sort-replace={{[^0-9\string\.\string\-]}{}}
]
```

The sort=double setting uses a double-precision floating point comparator.

The second sub-block contains all of the pictographs. I've now decided to order them according to the character code of the closest matching Unicode symbol. This isn't a problem for the card suits as the TEX parser library recognises the commands, but it doesn't (currently) have support for the marvosym commands. To handle them, I provide suitable definitions within `@preamble`. Using `\providecommand` will ensure these definitions don't override the marvosym definitions within the document. (Alternatively, write-preamble=false can be used to prevent `bib2gls` from writing the contents of `@preamble` to the `.glstex` file.)

For example, I can add the following to the `pictographs.bib` file:

```
@preamble{"
\providecommand{\Email}{\symbol{"1F584}}
\providecommand{\Letter}{\symbol{"1F582}}
\providecommand{\Mobilefone}{\symbol{"1F581}}
\providecommand{\Telefon}{\symbol{"1F57F}}"}
```

Alternatively, this could be added to another file called, say, `marvosym.bib` which can be loaded at the same time as `pictographs.bib`:

```
\GlsXtrLoadResources[
 selection=all,save-locations=false,
 src={marvosym,pictographs},
 symbol-sort-fallback=name,sort=letter-case
]
```

The sort=letter-case setting uses a case-sensitive character code comparison. Unlike the locale-sensitive sort methods, there's no attempt to detect word breaks and no characters, such as punctuation, are ignored.

This just leaves the terms and abbreviations:

```
\GlsXtrLoadResources[
 selection=all,save-locations=false,
 src={terms,abbreviations},
 entry-type-aliases={homograph=entry},
 symbol-sort-fallback=description,
 field-concat-sep={.},
 custom-sort-fallbacks={abbreviation=long+short,
  homograph=name+description}
]
```

This is basically the same as the previous example except that the `constants.bib` and `pictographs.bib` files are omitted.

The result is shown in Figure 11. If grouping is enabled (that is, if `bib2gls` is invoked with `--group` or `-g`) then all the numerical sort methods (such as sort=double) set the `group` label to `glsnumbers` which has the title given by the language-sensitive `\glsnumbersgroupname`. The character code sort methods (such as sort=letter-case) will assign the group based on the first character of the sort value. For the case-sensitive comparator, this can result in both lower and uppercase letter groups. Any character that isn't a letter (according to the Unicode specifications) is assigned to the group `glssymbols`, which has the title given by the language-sensitive `\glssymbolsgroupname`.

Alternatively you can force all entries in a given sub-block into a specific group with the group setting. For example:

```
\glsxtrsetgrouptitle{mcons}
  {Mathematical Constants}
\GlsXtrLoadResources[group={mcons},
 src={constants},⟨other settings⟩
]
\glsxtrsetgrouptitle{icons}{Pictographs}
\GlsXtrLoadResources[group={icons},
 src={pictographs},⟨other settings⟩
]
```

The sub-blocks can be reordered by simply rearranging the `\GlsXtrLoadResources` commands. (If you find yourself wanting to automatically order by sub-block title then you should actually be using a hierarchical glossary instead [6].)

The ability to provide `bib2gls` with commands in the `@preamble` makes it easier to adjust the sort value. For example, by providing the closest matching Unicode value for symbols (as above) or by providing a command that allows you to omit or reorder parts of the sort value. For example:

```
@preamble{"\providecommand{\sortart}[2]{#2}"}
@homograph{bravocry,name={bravo},
  description={\sortart{a}{cry of approval}}
}
```

This command would also need to be defined in the document:

```
\newcommand{\sortart}[2]{#1 #2}
```

So although it's not possible to programmatically define entries (like the earlier `\newconstant` command) the use of fallbacks, aliases and commands provided in the `@preamble` allows a flexible approach that can be customised on a per-document basis.

References

[1] D. Arseneau, M. Swift. The relsize package, 2013. ctan.org/pkg/relsize.

[2] M. Miklavec, T. Henlich, M. Vogel. The marvosym package, 2012. ctan.org/pkg/marvosym.

[3] W. Schmidt. The upgreek package, 2003. ctan.org/pkg/upgreek.

[4] N. Talbot. Sample `bib` files. dickimaw-books.com/latex/tugboat-bib2gls.

[5] N. Talbot. bib2gls: selection, cross-references and locations. *TUGboat* 41(3), 2020. tug.org/TUGboat/tb41-3/tb129talbot-bib2gls-more.pdf.

[6] N. Talbot. Logical glossary divisions (`type` vs `group` vs `parent`), 2020. dickimaw-books.com/gallery/?label=logicaldivisions.

[7] N. Talbot. bib2gls: Command line application to convert `.bib` files to `glossaries-extra.sty` resource files, 2020. ctan.org/pkg/bib2gls.

[8] N. Talbot. The glossaries-extra package, 2020. ctan.org/pkg/glossaries-extra.

[9] N. Talbot. The glossaries package, 2020. ctan.org/pkg/glossaries.

◇ Nicola L. C. Talbot
School of Computing Sciences
University of East Anglia
Norwich Research Park
Norwich NR4 7TJ
United Kingdom
https://www.dickimaw-books.com

A

$\zeta(3)$ (1.2020569) Apéry's constant

B

bravo cry of approval
bravo a hired ruffian or killer

C

♣ club
λ (1.30357) Conway's constant

D

diamond metastable allotrope of carbon
diamond four-sided shape with equal sides
diamond jubilee sixtieth anniversary
♢ diamond

E

e (2.71828) Euler's number
📧 email
✉ letter
γ (0.57721) Euler's constant

G

e^{π} (23.140692) Gelfond's constant
ϕ (1.61803) golden ratio

H

♡ heart
homographs
hypertext markup language (HTML) the standard markup language for creating web pages

I

i ($\sqrt{-1}$) imaginary unit

L

☎ telephone

M

mathematical markup language (MathML) markup language for describing mathematical notation

N

NASA National Aeronautics and Space Administration

O

Ofcom Office of Communications
Ofsted Office for Standards in Education
1 one

P

📱 mobile phone
π (3.14159) ratio of circumference of a circle to its diameter

R

$\sqrt{2}$ (1.41421) Pythagoras' constant

S

sample
♠ spade
scalable vector graphics (SVG) XML-based vector image format

X

extensible hypertext language (XHTML) XML version of HTML
extensible markup language (XML) a markup language that defines a set of rules for encoding documents

Z

0 zero

Figure 7: Default sorting (see p. 197)

Symbols

0 zero
1 one
$\sqrt{2}$ (1.41421) Pythagoras' constant

B

bravo a hired ruffian or killer
bravo cry of approval

C

♣ club

D

♢ diamond
diamond jubilee sixtieth anniversary
diamond four-sided shape with equal sides
diamond metastable allotrope of carbon

E

e (2.71828) Euler's number
📧 email
extensible hypertext language (XHTML) XML version of HTML
extensible markup language (XML) a markup language that defines a set of rules for encoding documents
e^{π} (23.140692) Gelfond's constant

H

♡ heart
homographs
hypertext markup language (HTML) the standard markup language for creating web pages

I

i ($\sqrt{-1}$) imaginary unit

L

✉ letter

M

mathematical markup language (MathML) markup language for describing mathematical notation
📱 mobile phone

N

NASA National Aeronautics and Space Administration

O

Ofsted Office for Standards in Education
Ofcom Office of Communications

S

sample
scalable vector graphics (SVG) XML-based vector image format
♠ spade

T

☎ telephone

Greek

γ (0.57721) Euler's constant
λ (1.30357) Conway's constant
π (3.14159) ratio of circumference of a circle to its diameter
ϕ (1.61803) golden ratio
$\zeta(3)$ (1.2020569) Apéry's constant

Figure 8: Sort suffix and fallbacks (see p. 198)

Symbols

γ (0.57721) Euler's constant
♣ club
♢ diamond
📧 email
♡ heart
✉ letter
📱 mobile phone
♠ spade
☎ telephone
$\zeta(3)$ (1.2020569) Apéry's constant
λ (1.30357) Conway's constant
$\sqrt{2}$ (1.41421) Pythagoras' constant
ϕ (1.61803) golden ratio
i ($\sqrt{-1}$) imaginary unit
e (2.71828) Euler's number
e^{π} (23.140692) Gelfond's constant
π (3.14159) ratio of circumference of a circle to its diameter

B

bravo a hired ruffian or killer
bravo cry of approval

D

diamond four-sided shape with equal sides
diamond metastable allotrope of carbon
diamond jubilee sixtieth anniversary

E

extensible hypertext language (XHTML) XML version of HTML
extensible markup language (XML) a markup language that defines a set of rules for encoding documents

H

homographs
hypertext markup language (HTML) the standard markup language for creating web pages

M

mathematical markup language (MathML) markup language for describing mathematical notation

N

NASA National Aeronautics and Space Administration

O

Ofcom Office of Communications
Ofsted Office for Standards in Education
1 one

S

sample
scalable vector graphics (SVG) XML-based vector image format

Z

0 zero

Figure 9: Sort suffix marker and custom fallbacks (see p. 199)

Symbols

γ (0.57721) Euler's constant
0 zero
ζ(3) (1.2020569) Apéry's constant
λ (1.30357) Conway's constant
$\sqrt{2}$ (1.41421) Pythagoras' constant
ϕ (1.61803) golden ratio
1 one
i ($\sqrt{-1}$) imaginary unit
e (2.71828) Euler's number
e^{π} (23.140692) Gelfond's constant
π (3.14159) ratio of circumference of a circle to its diameter

B

bravo a hired ruffian or killer
bravo cry of approval

C

♣ club

D

diamond four-sided shape with equal sides
diamond metastable allotrope of carbon
♢ diamond
diamond jubilee sixtieth anniversary

E

📧 email
extensible hypertext language (XHTML) XML version of HTML
extensible markup language (XML) a markup language that defines a set of rules for encoding documents

H

♡ heart
homographs
hypertext markup language (HTML) the standard markup language for creating web pages

L

✉ letter

M

mathematical markup language (MathML) markup language for describing mathematical notation
📱 mobile phone

N

NASA National Aeronautics and Space Administration

O

Ofcom Office of Communications
Ofsted Office for Standards in Education

S

sample
scalable vector graphics (SVG) XML-based vector image format
♠ spade

T

☎ telephone

Figure 10: Aliasing and concatenation (see p. 200)

Numbers

i ($\sqrt{-1}$) imaginary unit
0 zero
γ (0.57721) Euler's constant
1 one
ζ(3) (1.2020569) Apéry's constant
λ (1.30357) Conway's constant
$\sqrt{2}$ (1.41421) Pythagoras' constant
ϕ (1.61803) golden ratio
e (2.71828) Euler's number
π (3.14159) ratio of circumference of a circle to its diameter
e^{π} (23.140692) Gelfond's constant

Symbols

♠ spade
♡ heart
♢ diamond
♣ club
☎ telephone
📱 mobile phone
✉ letter
📧 email

B

bravo a hired ruffian or killer
bravo cry of approval

D

diamond four-sided shape with equal sides
diamond metastable allotrope of carbon
diamond jubilee sixtieth anniversary

E

extensible hypertext language (XHTML) XML version of HTML
extensible markup language (XML) a markup language that defines a set of rules for encoding documents

H

homographs
hypertext markup language (HTML) the standard markup language for creating web pages

M

mathematical markup language (MathML) markup language for describing mathematical notation

N

NASA National Aeronautics and Space Administration

O

Ofcom Office of Communications
Ofsted Office for Standards in Education

S

sample
scalable vector graphics (SVG) XML-based vector image format

Figure 11: Sub-blocks (see p. 201)