Aspell Developer's Manual

Kevin Atkinson (kevina@gnu.org)

This is the developer's manual for Aspell

This manual is designed for those who wish to develop Aspell. It is currently very sketchy. However, it should improve over time. The latest version of this document can be found at `http://savannah.gnu.org/download/aspell/manual/devel/devel.html`.

Table of Contents

1 Style Guidelines

As far as coding styles go I am really not that picky. The important thing is to stay consistent. However, please whatever you do, do not indent with more than 4 characters as I find indenting with more than that extremely difficult to read as most of the code ends up on the right side of the window.

2 How to Submit a Patch

Coming or removed soon.

3 C++ Standard Library

The C++ Standard library is not used directly except under very specific circumstances. The string class and the STL are used indirectly through wrapper classes and all I/O is done using the standard C library with light right helper classes to make using C I/O a bit more C++ like.

However the `new`, `new[]`, `delete` and `delete[]` operators are used to allocate memory when appropriate.

4 Templates

Templates are used in Aspell when there is a clear advantage to doing so. Whenever you use templates please use them carefully and try very hard not to create code bloat by generating a lot of unnecessary and duplicate code.

5 Error Handling

Exceptions are not used in Aspell as I find them more trouble than they are worth. Instead an alternate method of error handling is used which is based around the PosibErr class. PosibErr is a special Error handling device that will make sure that an error is properly handled. It is defined in `posib_err.hpp`. PosibErr is expected to be used as the return type of the function. It will automatically be converted to the "normal" return type however if the normal returned type is accessed and there is an "unhandled" error condition it will abort. It will also abort if the object is destroyed with an "unhandled" error condition. This includes ignoring the return type of a function returning an error condition. An error condition is handled by simply checking for the presence of an error, calling ignore, or taking ownership of the error.

The PosibErr class is used extensively throughout Aspell. Please refer to the Aspell source for examples of using PosibErr until better documentation is written.

6 Source Code Layout

'`common/`' Common code used by all parts of Aspell

'`lib/`' Library code used only by the actual Aspell library

'`data/`' Data files used by Aspell

'`modules/`'
: Aspell modules which are eventually meant to be pluggable

    '`speller/`'

        '`default/`'
        : Main speller Module.

    '`filter/`'

    '`tokenizer/`'

'`auto/`' Scripts and data files to automatically generate code used by Aspell

'`interface/`'
: Header files and such that external programs should use when in order to use the Aspell library.

    '`cc/`' The external C interface that programs should be using when they wish to use Aspell.

'`prog/`' Actual programs based on the Aspell library. The main Aspell utility is included here.

'`scripts/`'
: Miscellaneous scripts used by Aspell

'`manual/`'

'`examples/`'
: Example programs demonstrating the use of the Aspell library

7 Strings

7.1 String

The `String` class provided the same functionality of the C++ string except for fewer constructors. It also inherits `OStream` so that you can write to it with the `<<` operator. It is defined in `string.hpp`.

7.2 ParmString

ParmString is a special string class that is designed to be used as a parameter for a function that is expecting a string. It is defined in `parm_string.hpp`. It will allow either a `const char *` or `String` class to be passed in. It will automatically convert to a `const char *`. The string can also be accessed via the `str` method. Usage example:

```
void foo(ParmString s1, ParmString s2) {
   const char * str0 = s1;
   unsigned int size0 = s2.size()
   if (s1 == s2 || s2 == "bar") {
     ...
   }
}
...
String s1 = "...";
foo(s1);
const char * s2 = "...";
foo(s2);
```

This class should be used when a string is being passed in as a parameter. It is faster than using `const String &` (as that will create an unnecessary temporary when a `const char *` is passed in), and is less annoying than using `const char *` (as it doesn't require the `c_str()` method to be used when a `String` is passed in).

7.3 CharVector

A character vector is basically a `Vector<char>` but it has a few additional methods for dealing with strings which `Vector` does not provide. It, like `String`, is also inherits `OStream` so that you can write to it with the `<<` operator. It is defined in `char_vector.hpp`. Use it when ever you need a string which is guaranteed to be in a continuous block of memory which you can write to.

8 Smart Pointers

Smart pointers are used extensively in Aspell to simplify memory management tasks and to avoid memory leaks.

8.1 CopyPtr

The `CopyPtr` class makes a deep copy of an object whenever it is copied. The `CopyPtr` class is defined in `copy_ptr.hpp`. This header should be included wherever `CopyPtr` is used. The complete definition of the object `CopyPtr` is pointing to does not need to be defined at this point. The implementation is defined in `copy_ptr-t.hpp`. The implementation header file should be included at a point in your code where the class `CopyPtr` is pointing to is completely defined.

8.2 ClonePtr

`ClonePtr` is like copy pointer except the `clone()` method is used instead of the copy constructor to make copies of an object. If is defined in `clone_ptr.hpp` and implemented in `clone_ptr-t.hpp`.

8.3 StackPtr

A `StackPtr` is designed to be used whenever the only pointer to a new object allocated with `new` is on the stack. It is similar to the standard C++ `auto_ptr` but the semantics are a bit different. It is defined in `stack_ptr.hpp` — unlike `CopyPtr` or `ClonePtr` it is defined and implemented in this header file.

8.4 GenericCopyPtr

A generalized version of `CopyPtr` and `ClonePtr` which the two are based on. It is defined in `generic_copy_ptr.hpp` and implemented in `generic_copy_ptr-t.hpp`.

9 I/O

Aspell does not use C++ I/O classes and functions in any way since they do not provide a way to get at the underlying file number and can often be slower than the highly tuned C I/O functions found in the standard C library. However, some lightweight wrapper classes are provided so that standard C I/O can be used in a more C++ like way.

9.1 IStream/OStream

These two base classes mimic some of the functionally of the C++ functionally of the corresponding classes. They are defined in `istream.hpp` and `ostream.hpp` respectively. They are however based on standard C I/O and are not proper C++ streams.

9.2 FStream

Defined in `fstream.hpp`.

9.3 Standard Streams

`CIN`/`COUT`/`CERR`. Defined in `iostream.hpp`.

10 Config Class

The `Config` class is used to hold configuration information. It has a set of keys which it will accept. Inserting or even trying to look at a key that it does not know will produce an error. It is defined in `common/config.hpp`.

11 Filter Interface

11.1 Overview

In Aspell there are 5 types of filters:

1. *Decoders* which take input in some standard format such as iso8859-1 or UTF-8 and convert it into a string of `FilterChars`.
2. *Decoding filters* which manipulate a string of `FilterChars` by decoding the text is some way such as converting an SGML character into its Unicode value.
3. *True filters* which manipulate a string of `FilterChars` to make it more suitable for spell checking. These filters generally blank out text which should not be spell checked
4. *Encoding filters* which manipulate a string of `FilterChars` by encoding the text in some way such as converting certain Unicode characters to SGML characters.
5. *Encoders* which take a string of `FilterChars` and convert into a standard format such as iso8859-1 or UTF-8

Which types of filters are used depends on the situation

1. When *decoding words* for spell checking:
   - The *decoder* to convert from a standard format
   - The *decoding filter* to perform high level decoding if necessary
   - The *encoder* to convert into an internal format used by the speller module
2. When *checking a document*
   - The *decoder* to convert from a standard format
   - The *decoding filter* to perform high level decoding if necessary
   - A *true filter* to filter out parts of the document which should not be spell checked
   - The *encoder* to convert into an internal format used by the speller module
3. When *encoding words* such as those returned for suggestions:
   - The *decoder* to convert from the internal format used by the speller module
   - The *encoding filter* to perform high level encodings if necessary
   - The *encoder* to convert into a standard format

A `FilterChar` is a struct defined in `common/filter_char.hpp` which contains two members, a character, and a width. Its purpose is to keep track of the width of the character in the original format. This is important because when a misspelled word is found the exact location of the word needs to be returned to the application so that it can highlight it for the user. For example if the filters translated this:

```
Mr. foo said &quot;I hate my namme&quot;.
```

to this

```
Mr. foo said "I hate my namme".
```

without keeping track of the original width of the characters the application will likely highlight `e my ` as the misspelling because the spell checker will return 25 as the offset instead of 30. However with keeping track of the width using `FilterChar` the spell checker will know that the real position is 30 since the quote is really 6 characters wide. In particular the text will be annotated something like the following:

```
11111111111116111111111111111161
Mr. foo said "I hate my namme".
```

The standard *encoder* and *decoder* filters are defined in `common/convert.cpp`. There should generally not be any need to deal with them so they will not be discussed here. The other three filters, the *encoding filter*, the *true filter*, and the *decoding filter*, are all defined the exact same way; they are inherited from the `IndividualFilter` class.

11.2 Adding a New Filter

A new filter basically is added by placing the corresponding loadable object inside a directory reachable by Aspell via '`filter-path`' list. Further it is necessary that the corresponding filter description file is located in one of the directories listed by the '`option-path`' list.

The name of the loadable object has to conform to the following convention '`lib`*`filtername`*`-filter.so`' where *`filtername`* stands for the name of the filter which is passed to Aspell by the '`add-filter`' option. The same applies to the filter description file which has to conform to the following naming scheme: '*`filtername`*`-filter.opt`'.

To add a new loadable filter object create a new file.

Basically the file should be a C++ file and end in '`.cpp`'. The file should contain a new filter class inherited from `IndividualFilter` and a constructor function called '`new_`*`filtertype`*' (see [Constructor Function], page 6) returning a new filter object. Further it is necessary to manually generate the filter description file. Finally the resulting object has to be turned into a loadable filter object using libtool.

Alternatively a new filter may extend the functionality of an existing filter. In this case the new filter has to be derived form the corresponding valid filter class instead of the `IndividualFilter` class.

11.3 IndividualFilter class

All filters are required to inherit from the `IndividualFilter` class found in `indiv_filter.hpp`. See that file for more details and the other filter modules for examples of how it is used.

11.4 Constructor Function

After the class is created a function must be created which will return a new filter allocated with `new`. The function must have the following prototype:

```
C_EXPORT IndividualFilter * new_aspell_filtername_filtertype
```

Filters are defined in groups where each group contains an *encoding filter*, a *true filter*, and a *decoding filter* (see [Filter Overview], page 5). Only one of them is required to be defined, however they all need a separate constructor function.

11.5 Filter Description File

This file contains the description of a filter which is loaded by Aspell immediately when the '`add-filter`' option is invoked. If this file is missing Aspell will complain about it. It consists of lines containing comments which must be started by a `#` character and lines containing key value pairs describing the filter. Each file at least has to contain the following two lines in the given order.

```
ASPELL >=0.60
DESCRIPTION this is short filter description
```

The first non blank, non comment line has to contain the keyword `ASPELL` followed by the version of Aspell which the filter is usable with. To denote multiple Aspell versions the version number may be prefixed by '`<`', '`<=`', '`=`', '`>=`' or '`>`. If the range prefix is omitted '`=`' is assumed. The `DESCRIPTION` of the filter should be under 50, begin in lower case, and note include any trailing punctuation characters. The keyword `DESCRIPTION` may be abbreviated by `DESC`.

For each filter feature (see [Filter Overview], page 5) provided by the corresponding loadable object, the option file has to contain the following line:

```
STATIC filtertype
```

`filtertype` stands for one of `decoder`, `filter` or `encoder` denoting the entire filter type. This line allows to statically (see [Link Filters Static], page 9) link the filter into Aspell if requested by the user or by the system Aspell is built for.

```
OPTION newoption
DESCRIPTION this is a short description of newoption
TYPE bool
DEFAULT false
ENDOPTION
```

An option is added by a line containing the keyword `OPTION` followed by the name of the option. If this name is not prefixed by the name of the filter Aspell will implicitly do that. For the `DESCRIPTION` of a filter option the same holds as for the filter description. The `TYPE` of the option may be one of `bool`, `int`, `string` or `list`. If the `TYPE` is omitted `bool` is assumed. The default value(s) for an option is specified via `DEFAULT` (short `DEF`) followed by the desired `TYPE` dependent default value. The table [Filter Default Values], page 7 shows the possible values for each `TYPE`.

Type	Default	Available
bool	true	true false
int	0	any number value
string		any printable string
list		any comma separated list of strings

Table 1. Shows the default values Aspell assumes if option '`description`' lacks a `DEFAULT` or `DEF` line.

The `ENDOPTION` line may be omitted as it is assumed implicitly if a line containing `OPTION`, `STATIC`.

> **Note:** The keywords in a filter description file are case insensitive. The above examples use the all uppercase for better distinguishing them from values and comments. Further a filter description may contain blank lines to enhance their readability.

> **Note:** An option of `list` type may contain multiple consecutive lines for default values starting with `DEFAULT` or `DEF`, to specify numerous default values.

11.6 Retrieve Options by a Filter

An option always has to be retrieved by a filter using its full qualified name as the following example shows.

```
config->retrieve_bool("filter-filtername-newoption");
```

The prefix `filter-` allows user to relate option uniquely to the specific filter when *`filtername`*`-newoption` ambiguous an existing option of Aspell. The *`filtername`* stands for the name of the filter the option belongs to and *`-newoption`* is the name of the option as specified in the corresponding '`.opt`' file (see [Filter Description File], page 6

11.7 Compiling and Linking

See a good book on Unix programming on how to turn the filter source into a loadable object.

11.8 Programmer's Interface

A more convenient way recommended, if filter is added to Aspell standard distribution to build a new filter is provided by Aspell's programmers interface for filter. It is provided by the '`loadable-filter-API.hpp`' file. Including this file gives access to a collection of macros hiding nasty details about runtime construction of a filter and about filter debugging. Table [Interface Macros], page 9 shows the macros provided by the interface. For details upon the entire macros see '`loadable-filter-API.hpp`'. An example on how to use these macros can be found at '`examples/loadable/ccpp-context.hpp`' and '`examples/loadable/ccpp-context.cpp`'.

Macro	**Type**	**Description**	**Notes**
ACTIVATE_ENCODER	M	makes the entire encoding filter callable by Aspell	do not call inside class declaration; these macros define new_<filtertype> function;
ACTIVATE_DECODER	M	makes the entire decoding filter callable by Aspell	*as above*
ACTIVATE_FILTER	M	makes the entire filter callable by Aspell	*as above*
FDEBUGOPEN	D	Initialises the macros for debugging a filter and opens the debug file stream	These macros are only active if the `FILTER_PROGRESS_CONTROL` macro is defined and denotes the name of the file debug messages should be sent to. If debugging should go to Aspell standard debugging output (right now stderr) use empty string constant as filename
FDEBUGNOTOPEN	D	Same as "FDEBUGOPEN" but only if debug file stream was not opened yet	*as above*

FDEBUGCLOSE	D	closes the debugging device opened by “FDEBUGOPEN” and reverts it to “stderr”;	*as above*
FDEBUG	D	prints the filename and the line number it occurs	*as above*
FDEBUGPRINTF	D	special printf for debugging	*as above*

Table 2. Shows the macros provided by ‘`loadable-filter-API.hpp`’ (**M** mandatory, **D** debugging)

11.9 Adding a filter to Aspell standard distribution

Any filter which one day should be added to Aspell has to be built using the developer interface, described in [Programmer’s Interface], page 8. To add the filter the following steps have to be performed:

1. Decide whether the filter should be kept loadable if possible, or always be statically linked to Aspell.
2. Place the filter sources inside the entire directory of Aspell source tree. Right now use *$top_srcdir*`/modules/filter`.
3. Modify the ‘`Makefile.am`’ file on the topmost directory of the Aspell distribution. Follow the instructions given by the `#Filter Modules` section.
4. Run ‘`autoconf`’, ‘`automake`’, . . .
5. Reconfigure sources.
6. Clear away any remains of a previous build and rebuild sources.
7. Reinstall Aspell.
8. Test if filter has been added properly otherwise return to steps 2–7
9. Reconfigure sources with ‘`enable-static`’ flag and repeat steps 2–7 until your filter builds and runs properly in case of static linkage.
10. Add your source files to cvs, and commit all your changes. Or in case you are not allowed to commit to cvs submit a patch (see Chapter 2 [How to Submit a Patch], page 1) containing your changes.

12 Filter Modes

Filter modes are the preferred way to specify combinations of filters which are used regularly and thus abbreviate Aspell’s command line arguments.

A new filter mode is specified by a file named like the filter new mode and prefixed by .amf (Aspell Mode File). If such a file is accessible by the path set via filter-path option Aspell will try to load the contained mode specification.

12.1 Aspell Mode File

The first key in the made file has be the `mode` key. It is checked against the mode name part of the .amf file. If the `mode` key is missing mode file will be rejected.

The same holds for the `aspell` key which specifies the version(s) of Aspell which is(are) required by the filter.

If these two keys are followed by at least one `magic` key Aspell will be able to select the entire mode from extension and if required from contents of the file to spell implicitly.

The last key of the required keys is the `des[c[ription]]` key. It gives a short description of the filter mode which will displayed when type `aspell help`.

The rest of the file consists of the keys `filter` and `option` to load filters are set various options.

12.1.1 Version Line

Each version line must start with `aspell` and be followed by a version, optionally prefixed by a relational operator. The relation operator can be one of '<', '<=', '=', '>=' or '>' for allowing Aspell version with version number being lower, lower or equal, equal to, greater or equal or greater than required version number, respectfully. If the relation operator is omitted '=' is assumed.

12.1.2 Magic Line

The magic line contains a description which requirements files have to fulfill in order to implicitly activate the entire mode at least one such line is required. Each magic line has the following format:

```
MAGIC /<magic key>/<fileextention>[/<fileextention>]
```

The magic key consist of three ':' separated fields. The first two are byte counts the last is a regular expression. The first byte count indicates the first byte the regular expression will be applied to the second byte count indicates the number of bytes to test against the regular expression.

If mode selection should only occurred on basis of the listed file extensions the magic key should consist of the "<noregex>" special string.

At least one <fileextention> is required per MAGIC line. <fileextention> may not be empty and should not contain a leading '.' as this is assumed implicitly.

Multiple MAGIC lines are allowed. Modes may be extended limited by additional <label>.amf files located in –filter-path Thus file extensions may be prefixed by '+' or '-' to indicate that the entire extension has to be added ore removed from this <magic key> if neither is specified than a '+' is assumed implicitly.

12.1.3 Description Line

The required description line will be printed when typing `aspell help`. Keep it as short as possible. Possible abbreviations are `des` and `desc`.

12.1.4 Filter and Option Lines

The `filter` and `option` keys load filters and set filter options.

The value of the `filter` key is equal to the value of Aspell's `[add|rem]-filter` option.

Each `option` line has the following format:

```
OPTION <option> [<value>]
```

The format of the <option> and <value> is the same format as found in the Aspell configuration file.

13 Data Structures

Whenever possible you should try to use one of the data structures available. If the data structures do not provide enough functionality for your needs you should consider enhancing them rather than writing something from scratch.

13.1 Vector

The `vector` class is defined in `vector.hpp` and works the same way as the standard STL `vector` does except that it doesn't have as many constructors.

13.2 BasicList

`BasicList` is a simple list structure which can either be implemented as a singly or doubly linked list. It is defined in `basic_list.hpp`.

13.3 StringMap

`StringMap` is a associative array for strings. You should try to use this when ever possible to avoid code bloat. It is defined in `string_map.hpp`.

13.4 Hash Tables

Several hash tables are provided when `StringMap` is not appropriate. These hash tables provide a `hash_set`, `hash_multiset`, `hash_map` and `hash_multimap` which are very similar to SGI's STL implementation with a few exceptions. It is defined in `hash.hpp`.

13.5 BlockSList

`BlockSList` provided a pool of nodes which can be used for singly linked lists. It is defined in `block_slist.hpp`.

14 Mk-Src Script

A good deal of interface code is automatically generated by the '`mk-src.pl`' Perl script. I am doing it this way to avoid having to write a lot of relative code for the C++ interface. This should also make adding interface for other languages a lot less tedious and will allow the interface to automatically take advantage of new Aspell functionality as it is made available. The `mk-src.pl` script uses `mk-src.in` as its input.

14.1 mk-src.in

NOTE: This section may not always be up to date since it is manually converted from the pod source.

The format of '`mk-src.in`' is as follows:

```
The following characters are literals: { } / '\ ' \n = >

<items>
```

```
<items> := (<item>\n)+
<items> := <category>:\ <name> {\n<details>\n} | <<tab>><details>
<details> := <options>\n /\n <items>
<options> := (<option>\n)*
<option> := <key> [=> <value>]

<<tab>> means everything should be indented by one tab
```

See MkSrc::Info for a description of the categories and options

14.2 MkSrc::Info

`%info`

The info array contains information on how to process the info in 'mk-src.pl'. It has the following layout

```
<catagory> => options => []
              groups => [] # if undef than anything is accepted
              creates_type => "" # the object will create a new type
                                 # as specified
              proc => <impl type> => sub {}
```

where <impl type> is one of:

```
cc: for "aspell.h" header file
cxx: for C++ interface implemented on top of cc interface
native: for creation of header files used internally by aspell
impl: for defination of functions declared in cc interface.
      the definations use the native hedaer files
native_impl: for implementations of stuff declared in the native
              header files
```

each proc sub should take the following argv

```
$data: a subtree of $master_data
$accum:
```

<options> is one of:

```
desc: description of the object
prefix:
posib err: the method may return an error condition
c func:
const: the method is a const member
c only: only include in the external interface
c impl headers: extra headers that need to be included in the C impl
c impl: use this as the c impl instead of the default
cxx impl: use this as the cxx impl instead of the default
returns alt type: the constructor returns some type other than
  the object from which it is a member of
no native: do not attemt to create a native implementation
treat as object: treat as a object rather than a pointer
```

The `%info` structure is initialized as follows:

```
our %info =
(
 root => {
   options => [],
   groups => ['methods', 'group']},
 methods => {
   # methods is a collection of methods which will be inserted into
   # a class after some simple substation rules.  A $ will be
   # replaced with name of the class.
   options => ['strip', 'prefix', 'c impl headers'],
   groups => undef},
 group => {
   # a group is a colection of objects which should be grouped together
   # this generally means they will be in the same source file
   options => ['no native'],
   groups => ['enum', 'struct', 'union', 'func', 'class', 'errors']},
 enum => {
   # basic C enum
   options => ['desc', 'prefix'],
   creates_type => 'enum'},
 struct => {
   # basic c struct
   options => ['desc', 'treat as object'],
   groups => undef,
   creates_type => 'struct',},
 union => {
   # basic C union
   options => ['desc', 'treat as object'],
   groups => undef,
   creates_type => 'union'},
 class => {
   # C++ class
   options => ['c impl headers'],
   groups => undef,
   creates_type => 'class'},
 errors => {}, # possible errors
 method => {
   # A class method
   options => ['desc', 'posib err', 'c func', 'const',
               'c only', 'c impl', 'cxx impl'],
   groups => undef},
 constructor => {
   # A class constructor
   options => ['returns alt type', 'c impl', 'desc'],
   groups => 'types'},
 destructor => {
   # A class destructor
```

```
    options => [],
    groups => undef},
  );
```

In addition to the categories listed above a “methods” category by be specified in under the class category. A “methods” category is created for each methods group under the name “<methods name> methods”. When groups is undefined a type name may be specified in place of a category.

`%types`

types contains a master list of all types. This includes basic types and ones created in ‘`mk-src.in`’. The basic types include:

```
'void', 'bool', 'pointer', 'double',
'string', 'encoded string', 'string obj',
'char', 'unsigned char',
'short', 'unsigned short',
'int', 'unsigned int',
'long', 'unsigned long'
```

%methods

`%methods` is used for holding the “methods” information

14.3 MkSrc::Util

This module contains various useful utility functions:

`false` Returns 0.

`true` Returns 1.

`cmap EXPR LIST`
: Apply EXPR to each item in LIST and than concatenate the result into a string

`one_of STR LIST`
: Returns true if LIST contains at least one of STR.

`to_upper STR`
: Convert STR to all uppercase and substitute spaces with underscores.

`to_lower STR`
: Convert STR to all lowercase and substitute spaces with underscores.

`to_mixed STR`
: Convert STR to mixed case where each new word startes with a uppercase letter. For example "feed me" would become "FeedMe".

14.4 MkSrc::Read

`read` Read in ‘`mk-src.in`’ and return a data structure which has the following format:

```
      <tree>
      <tree> := <options>
                data => <tree>
    where each tree represents an entry in mk-src.in.
```

```
The following two options are always provided:
  name: the name of the entry
  type: the catagory or type name
Additional options are the same as specified in %info
```

14.5 MKSrc::Create

`create_cc_file` PARMS

Create a source file.

```
Required Parms: type, dir, name, data
Boolean Parms:  header, cxx
Optional Parms: namespace (required if cxx), pre_ext,
                accum
```

`create_file` FILENAME DATA

Writes DATA to FILENAME but only if DATA differs from the content of the file and the string:

```
Automatically generated file.
```

is present in the existing file if it already exists.

14.6 Code Generation Modes

The code generation modes are currently one of the following:

```
cc:     Mode used to create types suitable for C interface
cc_cxx: Like cc but typenames don't have a leading Aspell prefix
cxx:    Mode used to create types suitable for CXX interface
native: Mode in which types are suitable for the internal
        implementation
native_no_err: Like Native but with out PosibErr return types
```

14.7 MkSrc::CcHelper

Helper functions used by interface generation code:

```
to_c_return_type ITEM
    .

c_error_cond ITEM
    .
```

`make_func` NAME @TYPES PARMS ; %ACCUM

Creates a function prototype

Parms can be any of:

```
mode: code generation mode
```

`call_func` NAME @TYPES PARMS ; %ACCUM

Return a string to call a func. Will prefix the function with return if the functions returns a non-void type;

Parms can be any of:

```
mode: code generation mode
```

`to_type_name` ITEM PARMS ; %ACCUM
: Converts item into a type name.

Parms can be any of:

```
mode: code generation mode
use_type: include the actual type
use_name: include the name on the type
pos: either "return" or "other"
```

`make_desc` DESC ; LEVEL
: Make a C comment out of DESC optionally indenting it LEVEL spaces.

`make_c_method` CLASS ITEM PARMS ; %ACCUM
: Create the phototype for a C method which is really a function.

Parms is any of:

```
mode:       code generation mode
no_aspell:  if true do not include aspell in the name
this_name:  name for the parameter representing the
            current object
```

`call_c_method` CLASS ITEM PARMS ; %ACCUM
: Like make_c_method but instead returns the appropriate string to call the function. If the function returns a non-void type the string will be prefixed with a return statement.

`form_c_method` CLASS ITEM PARMS ; %ACCUM
: Like make_c_method except that it returns the array:

```
($func, $data, $parms, $accum)
```

which is suitable for passing into make_func. It will return an empty array if it can not make a method from ITEM.

`make_cxx_method` ITEM PARMS ; %ACCUM
: Create the phototype for a C++ method.

Parms is one of:

```
mode: code generation mode
```

Appendix A Copying

A.1 GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.