Docs/FlexCat_english

COLLABORATORS

	TITLE : Docs/FlexCat_english	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 14, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Docs/FlexCat_english

1.1 Docs/FlexCat_english.guide

```
                FlexCat V2.0 Documentation
**************************

   This file describes the Usage of FlexCat V2.0, a program which generates
catalogs and the source to handle them. FlexCat works similar to CatComp
and KitCat, but differs in generating any source you want. This is done by
using the so called Source descriptions, which are a template for the code
to generate. They can be edited and hence adapted to any programming
language and individual needs. (Hopefully!)


General:


                Disclaimer
                           Copyrights, (NO) warranty

                Survey
                               What is FlexCat?

                Installation
                         How can I get it working?

Using FlexCat:


                Program start
                        Calling FlexCat from the CLI

                Preferences
                          Changing FlexCat default behaviour

                Catalog description
                  Catalog description files (.cd-files)

                Catalog translation
                  Catalog translation files (.ct-files)
```

1.2 FlexCat_english.guide/Disclaimer

Copyright and other legal stuff
****

Jochen Wiedmann
Am Eisteich 9
72555 Metzingen
Deutschland

Since v1.8 program is developed by

Marcin Orlowski
ul. Radomska 38
71-002 Szczecin
Poland

carlos@amiga.com.pl
http://amiga.com.pl/flexcat/

software.

1.3 FlexCat_english.guide/Survey

```
                Survey
******
```

Since Workbench 2.1 the Amiga offers a rather pleasant system of using programs in different languages: The locale.library. (This is called localizing, that's what the name's for.)

The idea is simple: You select a language, the english in most cases and write your program in the same manner as you did without localizing, except that constant strings are replaced by certain function calls. Another function call makes it possible that the user selects another language when the program starts. (The latter function call loads an external file, the so called catalog and makes the former to read the strings from the catalog instead of using the predefined strings.)

These catalogs are independent from the program. All you need to do for adding another language is to create a new catalog file and this is possible at any time without changing the program.

But there are additional tasks for the programmer: He needs to create the catalogs, the predefined strings and some source to handle them all. (The functions that are mentioned above.) FlexCat is designed to make this in an easy and nearly automatic manner without losing flexibility especially in creating the source. An example should make this clear:

Lets assume that we want to write a HelloLocalWorld.c. Our final program will look like this:

```
#include <stdio.h>
#include <stdlib.h>
#include <HelloLocalWorld_Cat.h>  /*  You must include this! */

void main(int argc, char *argv[])
{
  printf("%s\n", msgHello);
}
```

Note that this is quite the same as the original HelloWorld.c except for replacing the string "Hello, world!" with a constant msgHello.

These constants and the related strings are defined in a so called Catalog description file. See

```
                Catalog description
```

. You always start by creating such a file called HelloLocalWorld.cd, which could look like this:

```
;   Comments are allowed, of course! Each line beginning with a
;   semicolon is assumed to be a comment
;
;   The language of the builtin strings:
#language english
;
```

```
;   The catalog version, used for a call to Locale/OpenCatalog().
;   This is different to Exec/OpenLibrary(): 0 means any catalog
;   version, other numbers must match exactly!
#version 0
;
;   This defines a string and the ID which allows to use it.
;   The number 4 says, that this string must not be shorter than
;   4 characters.
msgHello (/4/)
Hello, world!
```

By using FlexCat you create another two files from the catalog description: The include file HelloLocalWorld_Cat.h defines the constants and the HelloLocalWorld_Cat.c contains an array of strings and some initializing functions. You don't need to know what they do, just use them. Especially you don't need to know anything about the locale.library!

However, you might be interested, how these files look or even more, you might want to modify them. This is the difference between FlexCat and other catalog generators: With FlexCat you are not bound to a certain builtin format these files have. Instead it uses external template files, so called Source descriptions. This makes it possible, for example, to allow using catalogs with AmigaDOS 2.0. See
Source description
. If you use the
source descriptions from the FlexCat distribution you can create the source files with the following commands:

```
FlexCat HelloLocalWorld.cd HelloLocalWorld_Cat.c=C_c.sd
FlexCat HelloLocalWorld.cd HelloLocalWorld_Cat.h=C_h.sd
```

When your program is ready, you use FlexCat again to create so called Catalog translation files, one for each language you would like to support. (Except english, which is builtin.) See
Catalog translation
. Lets create a
german catalog translation:

```
FlexCat HelloLocalWorld.cd NEWCTFILE Deutsch.ct
```

This file would now look as follow:

```
## version
## language
## codeset 0
;   Comments ar eallowed, of course! Each line beginning with a
;   semicolon is assumed to be a comment
;
;   The language of the builtin strings:
;
;   The catalog version, used for a call to Locale/OpenCatalog().
;   This is different to Exec/OpenLibrary(): 0 means any catalog
;   version, other numbers must match exactly!
;
;   This defines a string and the ID which allows to use it.
;   The number 4 says, that this string must not be shorter than
;   4 characters.
msgHello

;Hello, world!
```

You see, it looks much like the catalog descriptions. FlexCat includes the comments from the catalog description, even where it is meaningless: Note the comment on the string length which shouldn't appear here as these informations must be in the catalog description only. All you have to do now is to fill in the informations on the version (a typical version string like $VER: Deutsch.catalog 1.1 (25.08.97) is expected), the language of the catalog translation (Deutsch for german here), the codeset (which should always be 0 for now, see Locale/OpenCatalog() for details) and of course the strings itself. FlexCat includes the original strings as comments, so you always know what to fill in. Finally you create the catalogs with commands like

```
FlexCat HelloLocalWorld.cd Deutsch.ct CATALOG Deutsch.catalog
```

Note, that you don't need the program itself or the source files created with FlexCat for the catalogs! You can create new catalogs at any time. It is usual to supply distributions with a file FlexCat.ct, so users can create own catalogs.

But what happens if you change the program later? Just edit the catalog description and use FlexCat to update the catalog translations:

```
FlexCat HelloLocalWorld.cd Deutsch.ct NEWCTFILE Deutsch.ct
```

All you need to do now is to enter new strings if needed.

1.4 FlexCat_english.guide/Installation

```
                Installation
************
```

FlexCat is written in pure Ansi-C (except for the localization), hence it should run on any Amiga and hopefully on other machines after recompiling. (The localizing is commented out in that case.) This holds for the created programs too: FlexCat is written using itself. All distributed source descriptions should create programs running on any Amiga and even any machine. (Of course you must ensure that the variable LocaleBase has the value NULL in the latter case.) Localizing, however, is possible beginning with Workbench 2.1 because the locale.library isn't available below.

It is not impossible to offer localizing without the locale.library: The source description files C_c_V20.sd and C_h_V20.sd give an example, where the iffparse.library is used to replace the locale.library, if it is not available. This gives Localizing for Workbench 2.0. See

```
                C
                .
```

Installing FlexCat is simple: Just copy the program to a directory in your search path and select a place for the source descriptions you need. (These are the files called something like xx_yy.sd, where xx is the programming language.) Probably you want to set the environment variable FLEXCAT.PREFS or FLEXCAT_SDDIR. See

```
                Program start
                .
```

If you want to use FlexCat in another language than the english you need to copy the respective catalog files too. E.g. for the german language copy the file Catalogs/Deutsch/FlexCat.catalog to Locale:Catalogs/Deutsch/ or to PROGDIR:Catalogs/Deutsch/ , where PROGDIR: is FlexCat's program directory. See

Using FlexCat source

.

1.5 FlexCat_english.guide/Program start

Calling FlexCat from the CLI

****

FlexCat is a CLI based program and doesn't operate from the workbench. It's calling syntax is

```
FlexCat CDFILE/A,CTFILE,CATALOG/K,NEWCTFILE/K,SOURCES/M,WARNCTGAPS/S,
        NOOPTIM/S,FILL/S,FLUSH/S,NOBEEP/S,NOLANGTOLOWER/S,NOBUFFEREDIO/S,
        MODIFIED/S,QUIET/S
```

Please note, that due to FlexCat portability, the argument parsing is not quite standard. Most notably, the only keywords you can (and must) specify are CATALOG and NEWCTFILE (those of type "/K"), others should be ommited, or be badly taken as argument itself. This is going to change probably in the next release.

Since v1.9, FlexCat implements simple preferences mechanism, which allows you to change default behaviour of FlexCat. See

Preferences

.

And now, the arguments meaning:

CDFILE

is the name of a catalog description to be read. This is always needed. Please note, that the base name of the source description is created from it making this case significant. See

Source description

.

CTFILE

is the name of a catalog translation file to be read. This is needed for creating catalogs or for updating an old catalog translation file using the NEWCTFILE argument: FlexCat reads the old file and the catalog description and creates a new catalog translation file containing the old strings and possibly some empty lines for new strings.

CATALOG

is the name of a catalog file to be created. This argument requires giving CTFILE as well.

NEWCTFILE

is the name of a catalog translation file to create. FlexCat reads strings from CTFILE, if this is given, strings missing in the catalog translation are replaced by empty lines. (The new catalog translation will contain only empty lines as strings, if CTFILE is omitted.)

SOURCES
are the names of source files to be created. These should be given in the form source=template where source is the file to create and template is the name of a source description file to be scanned.

If the source description isn't found, FlexCat tries to open a file with the same name in the directory PROGDIR:lib. (The subdirectory lib of the directory where the binary FlexCat itself lives.) You can overwrite this default with the environment variable FLEXCAT_SDDIR. Example:

```
FlexCat FlexCat.cd FlexCat_Cat.c=Templates/C_c_V20.sd
```

would look for a file Templates/C_c_V20.sd in the current directory first. If this wouldn't be found and no variable FLEXCAT_SDDIR would be present, FlexCat would look for PROGDIR:lib/Templates/C_c_V20.sd. But if FLEXCAT_SDDIR would exist and have the value Work:Flexcat, for example, then the existence of Work:FlexCat/Templates/C_c_V20.sd would be checked.

WARNCTGAPS
usually FlexCat doesn't warn about symbols missing in the catalog translation. This option will switch on such warnings.

NOOPTIM
Normally, if both strings (source in #?.cd file and translation in #?.ct one) are equall, FlexCat assumes there's no need to write it to the catalog file as it should be in program built-in string table already, from which it will be taken. But if you want, for some reasons these strings to be written (or in another words: if you want all strings to be written) use NOOPTIM.

FILL
This feature is highly useful for the translators. Normally, while working on the translation you got some strings still empty as you are working on them. But it obvious you want to check currently translated strings. Unfortunately all catalog creators including FlexCat write all empty strings too, which cause empty buttons or simmilar things to happen. Switch to forbid empty strings is not a good solution because prevents you from having such if you need. Also some bad written program may requre all strings to be in the catalog (even empty) e.g. due to lack of built-in strings. Using FILL option you force FlexCat to write source string (from #?.cd file) everytime it catch translation to be empty or be not present at all. NOTE: this is only for testing purposes. Final catalogs should always be created with no FILL swich used!

FLUSH
This switch is useful when you are translate and test your translation simultanously. As AmigaOS caches catalogs (as well as libraries, fonts, devices etc) in memory, you need to flush it (e.g. using C:AVAIL FLUSH command) every time you want new catalog to be reread from the disk (instead of using cached copy). If you specify this

```
     switch while creating the catalog, FlexCat will automatically flush
     all unused things from the memory. NOTE: FLUSH works only when you
     create new catalog. Otherwise it will be ignored.  Example:
              FlexCat Test.cd Test.ct CATALOG Test.catalog FLUSH

NOBEEP
     Since version 1.9, FlexCat will do DisplayBeep() to notice you about
     problems he encountered. Such behaviour is very useful when you call
     FlexCat from environment without standard output (e.g. you launch the
     script form the DOpus or other tool, etc). Of course you may don't
     like these beeps (however FlexCat is smart enough and beep only once,
     even you receive 20 warrnings, so don't be afraid of any
     beep-bombing). In such case use NOBEEP switch to shut FlexCat up.

NOLANGTOLOWER
     Normally, FlexCat makes #language entry argument (from #?.ct file)
     lowercased using utility.library call. Utility library calls
     locale.library if present, but I was reported that due to broken
     conversion table in some locales (czech for instance), it leads to
     incorrect strings. So this switch is the workaround for that problem.
     I strongly suggest to force your locale author to fix that bug, as
     some tools may also give you wrong results. And rememeber to keep
     #language name lowercased manually, if you need to use that switch
     (but don't use it unless really necessary).

NOBUFFEREDIO
     Buffered IO makes most applications often doing IO operations run
     faster.  So does FlexCat 2.0+. The speed up is mostly noticeable on
     systems using pooling devices (like (E)IDE), but the gain will also be
     reached on DMA bases systems (SCSI). FlexCat uses two 2KB buffers, so
     if you really think that's not the feature you like, that's the way to
     disable it.

MODIFIED
     This option tells FlexCat to compile the catalog only then, when the
     source #?.cd or #?.ct file were changed since last catalog creation.
     When catalog file is older than its sources, FlexCat just quits. This
     option is very useful when you want to create kind of shell scripts to
     process and compile more catalogs at once (e.g. for OS localisation or
     programs like DOpus5), and don't want to waste your time for
     recompilation of nonmodified catalogs.

QUIET
     Tells FlexCat to keep mouth shut unless really necessary.  It means
     that you won't see any warning messages.  Errors will be reported.

   For further examples of command lines see
               Survey
               .
```

1.6 FlexCat_english.guide/Preferences

```
                Changing default behaviour of FlexCat
************************************
```

Since version 1.9 FlexCat implements simple preference mechanism. By using environmental variable FLEXCAT.PREFS you can change program's default behaviour.

Variable FLEXCAT.PREFS is parsed using dos.library ReadArgs() call, thus all switches should be typed in one line with space as switch separator. The preferences template looks as follow:

```
SDDIR/K,NEW_MSG/K,WARNCTGAPS/S,NOOPTIM/S,FILL/S,FLUSH/S,NOBEEP/S,QUIET/S
```

NEW_MSG
can be used to customize the text, FlexCat uses to mark new strings apparing while updating the catalog translation file (using new description file and old translation). The default string is ***NEW***.

For detailed information about other tags, please read the

```
Program start
 chapter.
```

Note concerning SDDIR: while creating source file FlexCat firstly check the current dir, then directory set in preferences. If it still fails, it read FLEXCAT_SDDIR variable and finally the "PROGDIR:lib/" drawer. So using both preferences variable and FLEXCAT_SDDIR you can use two custom descriptors' drawers simultaneously.

1.7 FlexCat_english.guide/Catalog description

```
                Catalog description files
************************
```

A catalog description file contains four kinds of lines.

Comment lines
Any line beginning with a semicolon is assumed to be a comment line, hence ignored. (The string lines below are an exception. These may begin with a semicolon.)

Command lines
Any line beginning with a '#' (with the same exception as above) are assumed to be command lines. Possible commands are:

#language <str>
gives the programs default language, the language of the strings in the catalog description. Default is #language english.

#version <num>
gives the version number of catalogs to be opened. Note that this number must match exact and not be same or higher as in 'Exec/OpenLibrary'. An exception is the number 0, which accepts any catalog. Default is #version 0. See Locale/OpenCatalog for

```
      further information on catalog language and version.

   #lengthbytes <num>
      Instructs FlexCat to put the given number of bytes before a
      string containing its length. The length is the number of bytes
      in the string without length bytes and a trailing NUL byte.
      (Catalog files and hence catalog strings will have a trailing NUL
      byte. This is not always true for the default strings, depending
      on the source description file.) <num> must be between 0 and
      sizeof(long)=4, Default is #lengthbytes 0.

   #basename <str>
      Sets the basename of the source description. See

            Source description
            .  This overwrites the basename from the
      command line argument CDFILE.  See
            Program start
            .  Commands are
    case insensitive.

Description lines
    declare a string. They look like IDSTR (id/minlen/maxlen) where IDSTR
    is a identifier (a string consisting of the characters a-z,A-Z and
    0-9), id is a unique number (from now on called ID), minlen and maxlen
    are the strings minimum and maximum length, respectively. The latter
    three may be missing (but not the characters (//)!) in which case
    FlexCat chooses a number and makes no restrictions on the string
    length.  Better don't use the ID's, if you don't need. The lines
    following are the

String lines
    containing the string itself and nothing else. These may contain
    certain control characters beginning with a backslash:
   \b
      Backspace (Ascii 8)

   \c
      Control Sequence Introducer (Ascii 155)

   \e
      Escape (Ascii 27)

   \f
      Form Feed (Ascii 12)

   \g
      Display beep (Ascii 7)

   \n
      Line Feed, newline (Ascii 10)

   \r
      Carriage Return (Ascii 13)

   \t
      Tab (Ascii 9)
```

```
    \v
          Vertical tab (Ascii 11)

    \)
          The trailing bracket which is possibly needed as part of a (..)
          sequence, see
                Source description
                .

    \
          The backslash itself

    \xHH
          The character given by the ascii code HH, where HH are hex digits.

    \OOO
          The character given by the ascii code OOO, where OOO are octal
          digits.  Finally a single backslash at the end of the line causes
     concatening the following line. This makes it possible to use strings
     of any length, FlexCat makes no assumptions on string length.

   A string is hence given by a description line and the following string
line.  Let's see an example:
         msgHello (/4/)
         Hello, this is english!\n

The ID is missing here, so FlexCat chooses a suitable number. The number 4
instructs FlexCat, that the following string must not have less than four
characters and it may be of any length. See the file FlexCat.cd for a
further example.
```

1.8 FlexCat_english.guide/Catalog translation

```
                Catalog translation files
*************************

   Catalog translation files are very similar to catalog descriptions,
except for other commands and having no informations on string ID and
length.  (These are taken from the catalog description.) Any string from the
catalog description must be present (However, FlexCat omits writing strings
into the catalog which are identical to the default string.) and no
additional identifiers may occur. This is easy assured by using FlexCat to
create new catalog translation files. See
                Survey
                .

   The commands allowed in catalog translations are:
##version <str>
     Gives the catalog version as AmigaDOS version string. Example:
              ##version $VER: FlexCat.catalog 8.2 (25.08.97)
     The version number of this catalog is 8. Hence the catalog descriptions
     version number must be 0 or 8.
```

```
     You may replace the date string 27.09.93 with special keyword $TODAY.
     While creating catalog, $TODAY will be replaced by current date (note,
     only 1st occurance of $TODAY in $VER string will be processed).  If
     you want your version strings to always be recent type i.e.:
              $VER: FlexCat.catalog 3.1 (25.08.97)TODAY)

##rcsid $Date: 1997/10/01 13:06:53 $ $Revision: 1.12 $ $Id: FlexCat_english. ↩
   texinfo,v 1.12 1997/10/01 13:06:53 carlos Exp carlos $
     can be used in conjunction with a revision control system instead of
     ##version. <date> is the date in the form yy/mm/dd, time is the time
     (ignored), <rev> the revision and <name> the name to be used in the
     version string.

##name <name>
     is present for CatComp compatibility only. It replaces the <name>
     argument in the ##rcsid command.

##language <str>
     The catalogs language. Of course this should be another language than
     the catalog descriptions language. The ##language and ##version
     commands must be present in a catalog translation.

##codeset <num>
     Currently not used, must be 0. This is the default value.

## chunk <ID> <string>
     Adds a chunk ID to the catalog which consists if the given <string>.
     Usually one uses this to add comments to the catalog.
              ## chunk AUTH German catalog translation by Jochen Wiedmann

   The string from above looks like this in the catalog translation:
         msgHello
         Hallo, dies ist deutsch!\n

See Deutsch.ct as further example of a catalog translation.
```

1.9 FlexCat_english.guide/Source description

```
                Source description files
************************

   This is the special part of FlexCat. Until now there is nothing that
CatComp, KitCat and others don't offer too. The created source should make
it easy, to use the catalogs without losing flexibility. Any programming
language should be possible and any requirements should be satisfyable.
This seems like a contradiction, but FlexCat's solution are the source
description files containing a template of the source to be created. These
are editable as the catalog description and translation files are, hence
FlexCat can create any code.

   The source descriptions are searched for certain symbols which are
replaced by certain values. Possible symbols are the backslash characters
```

from above and additionally sequences beginning with a %. (This is well known for C programmers.)

%b
is the base name of the catalog description. See
Program start
.

%v
is the version number of the catalog description. Don't mix this up with the catalog version string from the catalog translation.

%l
is the catalog descriptions language. Please note, that this is inserted as a string. See %s below. below.

%n
is the number of strings in the catalog description.

%%
is the character % itself.

But the most important thing are the following seqences. These represent the catalog strings in different ways. Lines containing one or more of these symbols are repeated for any String.

%i
is the identifier from the catalog description.

%nd
%nx
%nc
is the strings ID in decimal, hexadecimal or octal characters, respectively. The number n tells FlexCat, how many characters the ID should use (the string will be filled with Zeros at the left). You may omit n: In this case the ID will take just the number of characters it needs.

%e
is the number of this string. Counting begins with 0.

%s
is the string itself; this will be inserted in a way depending on the programming language and can be controlled using the commands ##stringtype and ##shortstrings.

%na
is the string's ID. The difference between %na and e.g. %nx is that the %na produces string's ID parted to single bytes:
%2a in source descriptor will produce \x00\0x20
You may omit n. In this case the ID will take 4 bytes.

%nt
is the string's len. Please note, that the result value is always even.

%z
this item should be used together with %nt. Because %nt always returns even value having desriptor line like:

```
          static const char Block[] =
          {
               "%2a" "%2t" %s "%z"
          };
```

may lead to problems, especially while parsing such table, because %2t might be even while real string's %s lenght may be odd! So while parsing you read or skip one byte too much (I guess consequences are known). To avoid such problems %z was introduced. FlexCat replaces it with as many bytes (\x00) as many string's lenght lacks to even. So if string is 3 bytes long %nt returns 4 and %z adds one \x00

%(...)
inserts the text between the brackets for any string except the last. This is probably needed in Arrays, if the array entries should be separated by commas, but the last entry must not be followed by a comma. You can use %(,) in that case. Note that within the brackets there is no replacing of % sequences. Backslash sequences, however, are still allowed.

The control sequences %l and %s create strings. But how strings look depends on the program language. That's why the source description allows command lines similar to the catalog translation. These must begin with the first character of the line and any command must have its own line. Possible commands are:

##shortstrings
makes longer strings to be splitted on different lines. This is probably not always possible or not implemented into FlexCat and hence the default is to create one, probably very long string.

##stringtype <type>
Tells FlexCat how strings should look like. Possible types are

None
No additional characters are created. An image of the string is inserted and nothing else. No output of binary characters (the backslash sequences) is possible.

C
creates strings according to C. The strings are preceded and followed by the character ". Strings are splitted using the sequences "\ at the end of the line and " at the beginning of the new line. (The backslash is needed in macros.) Binary characters are inserted using \OOO. See
C
.

Oberon
is like string type C, except for the trailing backslash at the end of the line. See
Oberon
. This string type is recommended for Modula-2, too.

Assembler
Strings are created using dc.b. Readable ascii characters are preceded and followed by the character ', binary characters are inserted as $XX. See
Assembler

.

E
Strings are preceded and followed by the character '. A + concatenates strings which are spread on different lines. Binary characters are inserted like in C.

Let's look at an excerpt from the file C_h.sd creating an include file for the programming language C.

```
##stringtype C
##shortstrings

#ifndef %b_CAT_H    /*  Assure that this is read only once. */
#define %b_CAT_H


/*  Get other include files     */
#include <exec/types.h>
#include <libraries/locale.h>


/*  Prototypes  */
extern void Open%bCatalog(struct Locale *, STRPTR);
extern void Close%bCatalog(void);
extern STRPTR Get%bString(LONG);

/*  Definitions of the identifiers and their ID's           */
/*  This line will be repeated for any string.              */
#define %i %d

#endif
```

For the search path that is used for source descriptions see See

Program start
.

1.10 FlexCat_english.guide/Using FlexCat source

Including FlexCat source in own programs
****

Of course this depends on what source is created and hence on the source description. What we are talking here about are the source description files distributed with FlexCat. See
Source description
.

All source descriptions should allow using the program without locale.library. However, a global variable called LocaleBase (_LocaleBase for assembler) must be present and initialized with NULL or by a call to 'Exec/OpenLibrary'. No localizing is possible in the former case except when using the source description C_c_V20.sd. This allows localizing on

2.0 by repacing the locale.library with the iffparse.library. (A variable IFFParseBase has to be present for this and initialized like LocaleBase.) See

C

. The programmer does not need knowledge of these libraries ↩ except

when creating own source descriptions.

There are three functions and calling them is rather simple.

- : OpenCatalog (locale, language)
  This function possibly opens a catalog. The argument locale is a pointer to a Locale structure amd language is a string containing the name of the language that should be opened. In most cases these should both be NULL or NIL, respectively, because the user's defaults are overwritten otherwise. See 'Locale.OpenCatalog' for details.

  Non object oriented languages (C, Assembler, Modula) usually call these function OpenXXXCatalog, where XXX is the base name of the application: This allows to use different catalogs in the same program.

  If the user has Deutsch and Français as default languages and the programs base name is XXX this looks for the following files:

```
PROGDIR:Catalogs/Deutsch/XXX.catalog
LOCALE:Catalogs/Deutsch/XXX.catalog
PROGDIR:Catalogs/Français/XXX.catalog
LOCALE:Catalogs/Français/XXX.catalog
```

  where PROGDIR: is the programs current directory. (The order of PROGDIR: and LOCALE: can get changed in order to suppress a requester like Insert volume YYY.

  OpenCatalog is of type void (a procedure for Pascal programmers) and hence gives no result.

- : GetString (ID)
  Gives a pointer to the string with the given ID from the catalog description. Of course these strings are owned by locale.library and must not be modified.

  An example might be useful. Take the string from the catalog description example, which was called msgHello. The source descriptions declare a constant msgHello representing the ID. This could be printed in C using

```
printf("%s\n", GetString(msgHello));
```

- : CloseCatalog (void)
  This function frees the catalog (that is the allocated RAM) before terminating the program. You can call this function at any time even before OpenCatalog is called.

C

FlexCat source in C programs

C++

1.11 FlexCat_english.guide/C

```
                FlexCat source in C programs
============================
```

C source consists of two parts: A .c file which should be compiled and linked without further notice and an include file which should be included from any source part using catalog strings and which defines the ID's as macros.

The C compilers I know (SAS/C, Dice and gcc) allow automatic opening of libraries and initialization of the catalogs: Thus you need not call the functions OpenCatalog and CloseCatalog, your compiler does this for you. Similarly it calls the GetString functions for all catalog strings from within Opencatalog. This allows to simply write msgHello instead of GetString(msgHello).

If you define a preprocessor symbol LOCALIZE_V20 to the compiler (option -D LOCALIZE_V20 with gcc and Dice, DEF LOCALIZE_V20 with SAS/C), you get a program which can use catalogs under OS 2.0: The locale.library is replaced by the iffparse.library in that case. Your program needs an option like LANGUAGE Deutsch in that case: I function InitXXXCatalog (XXX being the base name of the application) should be called, if this option is present, which receives the language name as argument. This option is ignored, of course, if you have the locale.library. (It would be possible to do similar things under OS 1.3, but I don't want to support this obsolete version anymore.)

You loose a little bit functionality with this source description: For example, you cannot supply a Locale structure to OpenCatalog. However, 95% of all applications won't miss anything, others need to modify the source description.

For an example of a program using these source descriptions see
Survey
.

NOTE:

Since v1.9, distribution archive contains CatComp_h.sd source descriptor, which can be used with programs utilizing more than one catalog at the same time. Look inside to see how to update other source descriptors.

There're also another new source descriptor by Magnus Holmgren <lear@algonet.se>. The files Cat2h_c.sd and Cat2h_h.sd contains source descriptors that generates code similar to the one generated by Cat2h by Nico François (and also Cat2Inc by Magnus Holmgren ;). It uses a somewhat different approach to string handling, that is small and fast.

Rather than storing all string in an array, and scan that one each time (like CatComp normally does; there are ways around that though), the first two bytes of a string contains the ID. The "GetString" function, which takes a string as argument, then only reads these two bytes into a long word, and the string ID and default string is then known.

As of version 1.9, FlexCat is capable of generating that kind of output, using the %a command. The included files actually use %2a, and thus, only two ID bytes per string are generated (like Cat2h does). This should be enough for most applications. If you change the length, remember that the GetString() function need to be changed accordingly.

The generated header file defines all strings, and the source file contains code to open/close the catalog (with autoinit code for SAS/C and DICE), and a suitable GetString function. A quick look at the generated code should be enough to gather all the details, I think.

The code does currently not support multiple catalogs, nor change of version number and builtin language. Easy to add though (e.g. by using %b for all names (and references) needed to be unique e.g. Get%bString() etc), should the need arise.

1.12 FlexCat_english.guide/C++

FlexCat source in C++ programs
==========

Using FlexCat source in C++ programs is extremely comfortable: Almost everything is done by a special class implemented in the files C++_CatalogF.cc and C++_CatalogF.h. All you have to do is to rename these files into CatalogF.cc and CatalogF.h, compile them and create and compile two additional files using the source descriptions C++_cc.sd and C++_h.sd. The former will create a file with the strings (which must be compiled too, of course) and the latter will be included into your own program. A C++ program which uses FlexCat source will look like this:

```
#include <iostream.h>
extern "C"
{
#include <clib/exec_protos.h>
}
```

```
        #include "CatalogF.h"
        #include "HelloLocalWorld_Cat.h"

        struct LocaleBase *LocaleBase = 0;

        int main()
        { //  You must open the library here, even if your compiler supports
          //  Auto-Opening: This will usually break if the locale.library
          //  is not present. This is not what we want here as we just use
          //  the builtin strings in that case.
          LocaleBase = (struct LocaleBase *) OpenLibrary("locale.library", 38);

          const CatalogF cat(0, 0, HelloLocalWorld_ARGS);

          cout >> cat.GetString(msgHelloLocalWorld);

          if (LocaleBase)
              CloseLibrary(LocaleBase);
        }
```

A modification of gcc's libauto.a is available which will even allow to remove the lines concerning the variable LocaleBase.

1.13 FlexCat_english.guide/Oberon

```
                FlexCat source in Oberon programs
================================
```

There are different source descriptions: AmigaOberon.sd is designed for the current version of the AmigaOberon compiler, Oberon_V39.sd is for older versions and Oberon_V38.sd uses the Locale.mod from Hartmut Goebel. Oberon-A.sd is, of course for Oberon-A.

The function prototypes are

```
      XXX.OpenCatalog(loc: Locale.LocalePtr; language : ARRAY OF CHAR);
      XXX.GetString(num: LONGINT): Exec.StrPtr;
      XXX.CloseCatalog();
```

where XXX is the basename from the source description. See

```
                Source description
                .
```

Finally an example using FlexCat source:

```
      MODULE HelloLocalWorld;

      IMPORT  x:=HelloLocalWorld_Cat; Dos;

      BEGIN
        x.OpenCatalog(NIL, "");

        Dos.PrintF("%s\n", x.GetString(x.msgHello));

        (* Catalog will be closed automatically       *)
        (* when program exits.                        *)
```

```
        END Anything;
```

1.14 FlexCat_english.guide/Modula-2

```
                Flexcat source in Modula-2 programs
==================================

   Modula-2 supports a module concept similar to Oberon. This means that
the function names are always the same. Unlike Oberon, however, Modula-2
needs an implementation and a definition module, that's why you have to
create two files using the source descriptions Modula2Def.sd and
Modula2Mod.sd. These are adapted for the M2Amiga compiler. Note, that you
need the file OptLocaleL.def from version 4.3 of the M2Amiga compiler, too.

   The function prototypes are:
        PROCEDURE OpenCatalog(loc : ld.LocalePtr;
                              language : ARRAY OF CHAR);
        PROCEDURE CloseCatalog();
        PROCEDURE GetString(num : LONGINT) : ld.StrPtr;

where XXX is the base name from the source description.  See

                Source description
                .

   Finally an example of a program using FlexCat source:
        MODULE HelloLocalWorld;

        IMPORT hl: HelloLocalWorldLocale,
               io: InOut;

        BEGIN
          hl.OpenCatalog(NIL, "");

          io.WriteString(hl.GetString(hl.msgHello)); io.WriteLn;

          hl.CloseCatalog;
        END HelloLocalWorld.
```

1.15 FlexCat_english.guide/Assembler

```
                FlexCat source in Assembler programs
====================================

   Assembler source is created for usage with the Aztec Assembler.  This
should not be very different to other assemblers and you should be able to
implement own source descriptions. The source consists of two parts: A .asm
file which should be assembled and linked without further notice and an .i
```

include file which defines the string ID's and must be included by the using program.

The FlexCat-function names are slightly modified to allow the usage of different catalogs in one file: These are OpenXXXCatalog, CloseXXXCatalog and GetXXXString, where XXX is the base name from the source description. The concept is copied from the GadToolsBox and prooved good, as I think. See

Source description

.

As usual the function result is given in d0 and the functions save registers d2-d7 and a2-a7. OpenCatalog expects its arguments in a0 (pointer to Locale structure) and a1 (Pointer to language string) which should be NULL in most cases. GetString expects a pointer in a0. You should not care about what it points to.

Finally an example of a program using FLexCat source:

```
*   HelloLocalWorld.asm
        include "XXX.i" ; Opening this is a must. This
                        ; contains "xref OpenHelloLocalWorldCatalog", ...

        xref    _LVOOpenLibrary
        xref    _LVOCloseLibrary
        xref    _AbsExecBase

        dseg
LocNam: dc.b    "locale.library",0
        dc.l    _LocaleBase,4       ; Must be present under this name

        cseg

main:   move.l  #38,d0              ; Open locale.library
        lea     LocName,a1
        move.l  _AbsExecBase.a6
        jsr     _LVOOpenLibrary(a6)
*   NO exit, if OpenLibrary fails

        sub.l   a0,a0               ; Open catalog
        sub.l   a1,a1
        jsr     OpenHelloLocalWorldCatalog

        lea.l   msgHello,a0         ; Get pointer to string
        jsr     GetHelloLocalWorldString
        jsr     PrintD0             ; and print it

Ende:
        jsr     CloseHelloLocalWorldCatalog ; Close Catalog
        move.l  _LocaleBase,a1      ; Close locale.library
        move.l  a1,d0               ; this test is a must for 1.3
        beq     Ende1
        jsr     CloseLibrary
Ende1:
        rts
        end
```

1.16 FlexCat_english.guide/E

```
                FlexCat source in E programs
============================

   Since version 3.0 E allows to split a programs in separate modules. The
following describes the usage of E30b.sd which works with E3.0b or later.
(Version 3.0a had significant bugs, previous versions might use E21b.sd
which needs inserting the created source into the own source manually.)

   E30b.sd creates a module called Locale which contains a variable cat of
type catalog_XXX, where XXX is the basename from the source description.
See
                Source description
                .  A file HelloLocalWorld.e might look like this:
          MODULE '*Locale'
              -> Use this module

          DEF cat : PTR TO catalog_HelloLocalWorld
              -> This variable contains all the catalog strings and some
              -> methods. You must declare it in any module using
              -> Localization, but initialize it in the main module only.

          PROC main()

              localebase := OpenLibrary('locale.library', 0)
                  -> Open locale.library; No exit, if it cannot
                  -> be opened: We use the builtin strings in that case.

              NEW cat.create()
              cat.open()
                  -> As already mentioned, this is needed in the main
                  -> module only.

              WriteF('\s\n', cat.msg_Hello_world.getstr())
                  -> cat.msg_Hello_world one of the strings contained in
                  -> cat. This string declares a method getstr() which
                  -> reads the catalog and returns a pointer to the
                  -> localized string.

              cat.close()
              IF localebase THEN CloseLibrary(localebase)
          ENDPROC
```

1.17 FlexCat_english.guide/Appendix

```
Multiple catalogs support
=========================
```

Most of currently available source descriptors cannot be used for programs opening more than one catalog. In later releases it will surely change, and corrected source desriptors will be part of the release.

For now I supply the example of such source descriptor file. Read CatComp_h.sd to see how should the descriptor be defined to avoid multiple symols etc. In few words: use %b as prefix, suffix or other part of any name that is the vital part of source. If you table of strings is named STRING replace this by %b_STRINGS and you won't get dublicated lables any longer.

CatComp_h.sd produces source file similar to CatComp's used to generate, and can be used by those people who wish to use FlexCat but don't want to significantly change all of own programs.

1.18 FlexCat_english.guide/Future

Further development of FlexCat
****

However FlexCat seems to be almost finished, I got few items on my TODO list yet. And of course I'm open for suggestions, tips or critics. Especially I offer to include new string types because this is possible with very minor changes.

I would be pleased, if someone would send me new source descriptions and I could introduce them into further distributions. Any programming language, any extensions, provided that they are prooved good by testing the source in a real existing program. See See
Support
, for contact addresses.

1.19 FlexCat_english.guide/Support

FlexCat support sites
****

For software updates visit FlexCat's home page at:
http://amiga.com.pl/flexcat/

If you got any suggestion, bug report please e-mail me at:

carlos@amiga.com.pl

or via snail mail:

Marcin Orlowski
ul. Radomska 38

```
     71-002 Szczecin
     Poland
```

1.20 FlexCat_english.guide/Credits

```
Credits
*******

   Jochen Wiedmann's thanks go to:

Albert Weinert
     for KitCat, the predecessor of FlexCat which has done me valuable
     things, but finally wasn't flexible enough, and for the Oberon source
     descriptions.

Reinhard Spisser und Sebastiano Vigna
     for the Amiga version of texinfo. This documentation is written using
     it.

The Free Software Foundation
     for the original version of texinfo and many other excellent software.

Matt Dillon
     for DICE and especially for DME.

Alessandro Galassi
     for the italian catalog.

Lionel Vintenat
     for the E source description and its documentation, the french catalogs
     and bug reports.

Antonio Joaquín Gomez Gonzalez (u0868551@oboe.etsiig.uniovi.es) for
     the C++ source descripton, the spanish translation of the manual, the
     spanish catalog and the very good hint on speeding up the GetString
     function.

Olaf Peters (op@hb2.maus.de) for the Modula-2 source description
Russ Steffen (steffen@uwstout.edu)
     for the suggestion of the FLEXCAT_SDDIR variable.

Lauri Aalto (kilroy@tolsun.oulu.fi)
     for the finnish catalogs.

Marcin Orlowski (carlos@amiga.com.pl)
     for the polish catalogs and for maintaining the polish locale package.

Udo Schuermann (walrus@wam.umd.edu)
     for suggesting the WARNCTGAPS option and the ##chunk command.

Christian Hoj (cbh@vision.auc.dk)
     für die dänische Quelltextbeschreibung

The people of #AmigaGer
```

```
     for answering many stupid questions and lots of fun, for example
     stefanb (Stefan Becker), PowerStat (Kai Hoffmann), \ ill (Markus
     Illenseer), Quarvon (Jürgen Lang), ZZA (Bernhard Möllemann), Tron
     (Mathias Scheler), mungo (Ignatios Souvlatzis), \ jow (Jürgen
     Weinelt) und Stargazer (Petra Zeidler).

Commodore
     for the Amiga and Kickstart 2.0. Keep on developing it and I'll be an
     Amiga-user for the next 8 years too. ;-)

   Marcin's thanks go to:

Jochen Wiedmann for creating FlexCat
Magnus Holmgren <lear@algonet.se> for additional source descriptor Cat2h
Members of Amiga Translators' Organization <http://ato.vapor.com/ato/>
     for creating additional translations and updating existing ones:

    Serbian catalog file by Ljubomir Jankovic <lurch@afrodita.rcub.bg.ac.yu>
    Czech translation by Vit Sindlar <xsindl00@stud.fee.vutbr.cz>
    Svedish translation by Magnus Holmgren <lear@algonet.se> and Hjalmar Wikholm < ↩
       hjalle@canit.se>
    Finnish translation updated by Mika Lundell <c71829@uwasa.fi>
    Italian translation reworked by Luca Nora <ln546991@silab.dsi.unimi.it> and  ↩
       Giovanni Addabbo <gaddabbo@imar.net>
```

1.21 FlexCat_english.guide/History

```
History of development
**********************

   The history of FlexCat development is logged in the file FlexCat.history,
which is integral part of the distribution archive.
```

1.22 FlexCat_english.guide/Index

```
                Index
*****


                .cd
                                                Catalog description

                .ct
                                                Catalog translation

                .sd
                                                Source description

                Adress
                                             Disclaimer
```

AmigaOberon
Oberon

Ascii-Code
Catalog description

Assembler
Assembler

Author
Disclaimer

AutoC_c.sd
C

AutoC_h.sd
C

AztecAs_asm.sd
Assembler

AztecAs_i.sd
Assembler

C
C

C++
C++

C++_CatalogF.cc
C++

C++_CatalogF.h
C++

C++_cc.sd
C++

C++_h.sd
C++

Cat2h_c.sd
C

Cat2h_h.sd
C

CATALOG
Program start

Catalog description
Catalog description

Catalog translation
Catalog translation