FlexCat

The flexible catalog generator

Version 1.4

Jochen Wiedmann

Am Eisteich 9
72555 Metzingen (Deutschland)
Tel. 07123 / 14881
Internet: wiedmann@uni-tuebingen.de

1 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", GetString(msgHello));
}
```

Note that this is quite the same as the original '`HelloWorld.c`' except for replacing the string "Hello, world!" by a function call.

The above program uses a constant '`msgHello`'. A call to the functon `GetString` replaces this by the respective string. These constants and strings are defined in a so called `Catalog description` file. (see Chapter 4 [Catalog description], page 5. You always start by creating such a file called '`HelloLocalWorld.cd`', which could look like this:

```
; Comments ar eallowed, 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 the `GetString` function. You don't need to know how this look, 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 them changed. This is the difference between FlexCat and other catalog generators: FlexCat is not forced to use a special builtin format for creating these files. 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 Chapter 6 [Source description], page 8. 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=AutoC_c.sd'
'FlexCat HelloLocalWorld.cd HelloLocalWorld_Cat.h=AutoC_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 Chapter 5 [Catalog translation], page 7. Lets create a german catalog translation:

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

This file would no look as follows:

```
## 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 1.0 (11.03.94)`' 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 NewCatalog.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.

2 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 Section 7.1 [C], page 11.

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_SDDIR*. See Chapter 3 [Program start], page 4.

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/FlexCat.catalog`' or to '`PROGDIR:Catalogs/Deutsch/FlexCat.catalog`', where '`PROGDIR:`' is FlexCat's program directory. See Chapter 7 [Using FlexCat source], page 10.

3 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
```

where the arguments mean

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 Chapter 6 [Source description], page 8.

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 CDFILE 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 shoud 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.

For further examples of command lines see Chapter 1 [Survey], page 1.

4 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 Chapter 6 [Source description], page 8. This overwrites the basename from the command line argument CDFILE. See Chapter 3 [Program start], page 4.

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 Chapter 6 [Source description], page 8.

'`\\`' 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.

5 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 Chapter 1 [Survey], page 1.

The commands allowed in catalog translations are:

`##version <str>`

Gives the catalog version as AmigaDOS version string. Example:

```
'##version $VER: Deutsch.ct 8.1 (27.09.93)'
```

The version number of this catalog is 8. Hence the catalog descriptions version number must be 0 or 8.

`##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.

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.

6 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 Chapter 3 [Program start], page 4.

'`%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.

'`%d`' is the strings ID.

'`%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`'.

'`%(...)`' 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 Section 7.1 [C], page 11.

Oberon is like string type C, except for the trailing backslash at the end of the line. See Section 7.3 [Oberon], page 13. 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 Section 7.5 [Assembler], page 15.

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 Chapter 3 [Program start], page 4.

7 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 Chapter 6 [Source description], page 8.

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 Section 7.1 [C], page 11. 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.

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.

7.1 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.

Three different versions are available: '`AutoC_c.sd`' is the the simplest. They use the autoinizializing possibilities of `Dice` and `SAS/C` and are thus for those compilers only. All you have to do here is using the `GetString` function. Unfortunately you don't have all possibilities with these source descriptions, at least without own extensions, but this should be sufficient for 95% of all applications. Together with '`AutoC_c.sd`' you need the file '`AutoC_h.sd`' which creates the include files. For an example of a program using these source descriptions see Chapter 1 [Survey], page 1.

'`C_c_V21.sd`' and '`C_h.sd`' are what you need if you either

1. have another compiler than Dice or SAS/C
2. want to use all possibilities of the `locale.library/OpenCatalogA` call or
3. want to use more than one catalog file

The last reason is why the function names are slightly modified to `OpenXXXCatalog`, `GetXXXString` and `CloseXXXCatalog` where '`XXX`' is the base name from the source description. See Chapter 6 [Source description], page 8. These are the function prototypes:

```
void OpenXXXCatalog(struct Locale *loc, char *language);
STRPTR GetXXXString(ULONG);
void CloseXXXCatalog(void);
```

Using these source descriptions you have to do the initialization and termination manually. A program using these sourcce descriptions looks like this:

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

struct Library *LocaleBase;

void main(int argc, char *argv[])
{ /*  Open the library for yourself, even if the compiler supports  */
  /*  automatic opening. NO exit, if OpenLibrary fails:  */
  /*  We use the builtin strings in that case.  */
  /*  For the same reason we must not use autoinitialization of the */
  /*  variable LocaleBase, even if our compiler supports it!  */
  LocaleBase = OpenLibrary("locale.library", 38);

  OpenHelloLocalWorldCatalog(NULL, NULL);

  printf("%s\n", GetHelloLocalWorldString(msgHello));

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

The last version, being the source descriptions '`C_c_V20.h`' and '`C_h.sd`', is functionally identical except for supporting catalogs under 2.0. This is done by using the `iffparse.library` and reading the catalogs manually.[1] Programs using this look quite the same as the example above except for an option LANGUAGE which should be used under 2.0 only for calling `OpenXXXCatalog` and probably even be disabled under 2.1+.

7.2 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:

[1] Of course you could do this even without the `iffparse.library` and thus have catalogs under 1.3. However, I don't want to support 1.3 anymore.

```
#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`.

7.3 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 Chapter 6 [Source description], page 8.

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;
```

7.4 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 Chapter 6 [Source description], page 8.

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.
```

7.5 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 Chapter 6 [Source description], page 8.

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
```

7.6 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 Chapter 6 [Source description], page 8. 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
```

Further development of FlexCat

I don't expect much further development for I think FlexCat to be rather complete. 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. And I would appreciate receiving new catalogs. It is enough to insert the strings in the file '`NewCatalogs.ct`' which is part of the distribution.

Credits

My 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@felix.univ.szczecin.pl)
: for the polnish catalogs and for maintaining the polnish locale package.

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. ;-)

Index

•

.cd ... 5
.ct ... 7
.sd ... 8

A

AmigaOberon ... 13
Ascii-Code ... 6
Assembler ... 15
AutoC_c.sd ... 11
AutoC_h.sd ... 11
AztecAs_asm.sd ... 15
AztecAs_i.sd ... 15

C

C ... 11
C_c_V20.sd ... 11
C_c_V21.sd ... 11
C_h.sd ... 11
C++ ... 12
C++_CatalogF.cc ... 12
C++_CatalogF.h ... 12
C++_cc.sd ... 12
C++_h.sd ... 12
Catalog description ... 5
Catalog translation ... 7
CLI ... 4
Contributions ... 17
Control characters ... 6
Credits ... 17

D

Deutsch.ct ... 7

E

E ... 16
E21b.sd ... 16
E30b.sd ... 16

F

FlexCat ... 17
FlexCat source ... 10
FlexCat.cd ... 7
Future ... 17

I

Installation ... 3

M

Modula-2 ... 14
Modula2Def.sd ... 14
Modula2Mod.sd ... 14

O

Oberon ... 13
Oberon-A ... 13

Oberon_V38.sd 13
Oberon_V39.sd 13

R

Requirements 3

S

Source description 8
Survey 1

U

Using FlexCat source 10

W

Workbench 4

Table of Contents

1 Survey **1**

2 Installation **3**

3 Calling FlexCat from the CLI **4**

4 Catalog description files **5**

5 Catalog translation files **7**

6 Source description files **8**

7 Including FlexCat source in own programs **10**

7.1 FlexCat source in C programs 11
7.2 FlexCat source in C++ programs 12
7.3 FlexCat source in Oberon programs 13
7.4 Flexcat source in Modula-2 programs 14
7.5 FlexCat source in Assembler programs 14
7.6 FlexCat source in E programs 16

Further development of FlexCat **16**

Credits **17**

Index **18**