FlexCat

The flexible catalog generator

Version 1.0

Jochen Wiedmann

Am Eisteich 9
72555 Metzingen (Deutschland)
Tel. 07123 / 14881
Internet: wiedmann@mailserv.zdv.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. Using catalogs with FlexCat works like this:

1. You start by creating a file containing the predefined strings and and their description: The `catalog description`. The description consists of the string itself, an identifier which is used in the program in place of the string and informations on the strings minimal and maximal length. See Chapter 4 [Catalog description], page 3.
2. A `source description` file is used to create source from the catalog description: This is a template file containing certain patterns which are substituted with predefined values, for example the strings. This source description can be edited and hence adapted to any programming language and any individual needs. Some source descriptions (Assembler, Oberon and C, the latter allowing localizing under 2.0 or 2.1 and above) are already part of the FlexCat distribution. These can be used as examples or as they are without any further thinking. See Chapter 6 [Source description], page 6.
3. Using FlexCat you create a so called `catalog translation` file for every additional language. These are very similar to the catalog description except that they contain empty lines instead of the strings and miss the string description which is still part of the catalog description. All you need to do is replace the empty lines with the respective strings in the new language. FlexCat makes this easy by adding the original strings as comment lines, hence it is clear, what words you should insert. Finally FlexCat is used again to create a `catalog` file from the catalog description and the catalog translation. (This can be done at any time, even if the program is finished without any changes.) See Chapter 5 [Catalog translation], page 5.

2 Installation

FlexCat is written in pure Ansii-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 9.

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.) If you want to use FlexCat in another language than the english you need to copy the respective catalog files too. For the german language for example (actually this is the only one available) 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 8.

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 not, that the base name of the source description is created from it end hence this is case significant. See Chapter 6 [Source description], page 6.

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.

An example:

```
'FlexCat Prog.cd NEWCTFILE NewCatalog.ct prog_cat.c=C_c_V21.sd'
```

reads the catalog description '`Prog.cd`' and creates the catalog translation '`NewCatalog.ct`' and the source file '`prog_cat.c`' from it. The latter is created using the source description file '`C_c_V21.sd`'. The base name of the source description is '`Prog`'. (Please not the uppercase letter!)

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 and 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 be exact and not 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 lower or equal sizeof(long), that is lower or equal four. Default is '`#lengthbytes 0`'.

`#basename <str>`

Sets the basename of the source description. See Chapter 6 [Source description], page 6. This overwrites the basename from the command line argument CDFILE. See Chapter 3 [Program start], page 2.

Commands are case insensitive.

Description lines

declare a string. They look like '`IDSTR (id/minlen/maxlen)`' where '`IDSTR`' is a identifier (a string constisting of the characters a-z,A-Z and 0-9), '`id`' is a unic number (from now on called as 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 6.

'`\\`' 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 that for other commands and having no informations on string ID and length. (These are taken from the catalog description.) Of 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 3 [Program start], page 2.

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 loosing flexibility. Any programming language should be possible and any requirements should be satisfyable. This seems like a cantradiction, 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 catalog 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 2.

'`%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.

'`%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 9.

Oberon is like string type C, except for the trailing backslash at the end of the line.

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 7.3 [Assembler], page 11.

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
```

7 Including FlexCat source in own programs

Of course this depends on how the 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 6.

All source descriptions should allow using the program without `locale.library`. However, a globale variable called '`LocaleBase`' ('`_LocaleBase`' for assembler) must be present and ininitialized 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 '`LocaleBase`'.) See Section 7.1 [C], page 9. 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 '`Francais`' 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/Francais/XXX.catalog'
'LOCALE:Catalogs/Francais/XXX.catalog'
```

where '`PROGDIR:`' is the programs current directory. (The order of '`PROGDIR:`' and '`LOCALE:`' can be changed 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 IS's as macros using '`#define`'.

Two different versions are available for the '`.c`' part: '`C_c_V21.sd`' is a rather simple version using the respective functions of the `locale.library` and allowing localizing beginning with Workbench 2.1. But '`C_c_V20.sd`' replaces the `locale.library` with the `iffparse.library` if the former isn't available and the latter is. This allows localizing for Workbench 2.0 too. Programs using this should have an option `Language` and give the corresponding argument to '`OpenCatalog`'. This option should not be used in 2.1 and above and hence the language argument of '`OpenCatalog`' should still be '`NULL`'.

Of course it would be possible to write a third version using catalogs with Ansii C, but I don't want to support 1.3 anymore.

To separate the FlexCat functions '`OpenCatalog`' and '`CloseCatalog`' from the corresponding `Locale` functions with the same names and to allow different catalogs in one program the FlexCat functions get slightly modified names here: '`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 6.

The function prototypes are:

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

Finally an example of a program using FlexCat:

```
#include <stdio.h>
#include <stdlib.h>
#include <XXX.h>     /* Including this is a must!  */
#include <clib/exec_protos.h>

/* Open the library for yourself, even if the compiler */
/* supports automatic opening.  */
struct Library *LocaleBase;

void main(int argc, char *argv[])
```

```
{
  LocaleBase = OpenLibrary("locale.library", 38);
  /*  NO exit, if OpenLibrary fails and you don't need      */
  /*  Locale functions elsewhere.  */
  OpenXXXCatalog(NULL, NULL);

  ...   /* other functions      */

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

  ...   /* other functions again    */

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

7.2 FlexCat source in Oberon programs

There are two different source descriptions: '`Oberon_V38.sd`' creates source using '`Locale.mod`' from Hartmut Goebel. '`Oberon_V39.sd`' creates source using the '`Locale.mod`' distributed with `AmigaOberon`.

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 6.

Finally an example using FlexCat source:

```
MODULE Anything;

IMPORT  x:=XXX; Dos;

BEGIN
  x.OpenCatalog(NIL, "");

  ... (* Other functions    *)

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

  ... (* other functions again                      *)
(* Catalog will be closed automatically       *)
(* when program exits.                        *)
  END Anything;
```

7.3 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 6.

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 an string ID in d0.

Finally an example of a program using FLexCat source:

```
include "XXX.i" ; Opening this is a must. This
; contains "xref OpenXXXCatalog",...

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 and you don't need Locale
*   functions elsewhere

move.l #0,a0     ; Open catalog
move.l #0,a1
jsr OpenXXXCatalog

...      ; other functions

move.l #msgHello,d0     ; Get pointer to string
jsr GetXXXString
jsr PrintD0      ; and print it
```

```
...      ; other functions again

Ende:
jsr CloseXXXCatalog     ; 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
```

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 depending 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.

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.

The people of #AmigaGer

for answering many stupid questions and lots of fun, for example PowerStat (Kai Hoffmann), ZZA (Bernhard Moellemann), Stargazer (Petra Zeidler), stefanb (Stefan Becker), Tron (Mathias Scheler) and ill (Markus Illenberger).

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

Table of Contents