FlexCat

Der flexible Kataloggenerator

Version 1.0

Jochen Wiedmann

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

1 Übersicht

Seit der Workbench 2.1 bietet der Amiga ein sehr schönes System an, mit dem Programme in verschiedenen, praktisch beliebigen Sprachen benutzt werden können: Die `locale.library`. (Man nennt diesen Vorgang `Lokalisierung`, daher der Name.)

Die Idee ist eigentlich recht simpel: Man wählt eine Sprache, meist die englische aus und schreibt sein Programm ganz normal, abgesehen davon, daß Strings nicht mehr direkt eingegeben werden, sondern über einen Funktionsaufruf im Programm verwendet werden. Durch einen weiteren Funktionsaufruf zu Beginn des Programms erhält der Benutzer nun die Möglichkeit, anstelle der vorgegebenen Strings andere zu wählen, die in einer externen Datei, einem sogenannten `Katalog` enthalten sind.

Diese Katalogdateien sind vom Programm unabhängig. Möchte man das Programm in einer weiteren Sprache betreiben, so ist lediglich eine neue Katalogdatei zu erzeugen, das eigentliche Programm muß nicht geändert werden.

Auf den Programmierer kommen dadurch aber zusätzliche Aufgaben hinzu: Es müssen die Kataloge erzeugt werden, die Strings nach wie vor eingegeben werden und es muß zusätzlicher Code erzeugt werden, der die Behandlung der Kataloge übernimmt. Dies soll durch FlexCat so weit wie möglich vereinfacht und automatisiert werden, ohne dabei auf Flexibilität (vor allem in Bezug auf den erzeugten Quelltext) zu verzichten. Mit FlexCat geht das Ganze folgendermaßen vor sich:

1. Es wird zunächst eine Datei erzeugt, in der die vorgegebenen Strings enthalten und beschrieben sind: Die `Katalogbeschreibungsdatei`. Die Beschreibung enthält neben den Strings z.B. Angaben über minimale und maximale Länge. Siehe Abschnitt 4 [Catalog description], Seite 3.
2. Mit Hilfe einer `Quelltextbeschreibungsdatei` wird aus der Katalogbeschreibung Quelltext erzeugt: Die Quelltextbeschreibung ist praktisch eine Vorlage, in der FlexCat gewisse Symbole durch vorgegebene Werte, z.B. die Strings aus der Katalogbeschreibung ersetzt. Da die Quelltextbeschreibungsdatei mit einem Editor bearbeitet werden kann, kann der Quelltext an beliebige Programmiersprachen und für beliebige Anforderungen angepaßt werden. Gewisse Vorlagen (in Assembler, Oberon und C, letzteres wahlweise mit Lokalisierung unter 2.0 oder erst ab 2.1) sind bereits vorhanden. Sie können entweder als Beispiele dienen oder einfach verwendet werden, ohne sich über den erzeugten Quelltext den Kopf zu zerbrechen. Siehe Abschnitt 6 [Source description], Seite 6.
3. Mit Hilfe von FlexCat kann man dann für jede zusätzliche Sprache eine `Katalogüber\-setzungs\-datei` erzeugen. Diese ähnelt der Katalogbeschreibungsdatei, enthält aber für jeden String eine leere Zeile. In diese müssen nun die entsprechenden Strings in der neuen Sprache eingetragen werden. Damit dies möglichst einfach geht, folgen auf die leeren Zeilen jeweils Kommentarzeilen, die die ursprünglichen Strings enthalten. Es ist dadurch immer klar, welcher String wo einzusetzen ist. Aus der Katalogbeschreibungs- und der -übersetzungsdatei kann FlexCat dann anschließend einen Katalog erzeugen. (Dieser Vorgang kann zu beliebiger Zeit erfolgen, auch wenn das Programm längst fertig ist.) Siehe Abschnitt 5 [Catalog translation], Seite 5.

2 Installation des Programms

FlexCat ist in Ansi-C geschrieben und sollte daher auf jedem Amiga und nach evtl. Neucompilierung sogar auf jedem anderen Rechner laufen. Das gilt ebenso für die erzeugten Programme, denn FlexCat ist mit sich selbst erzeugt worden. Die mitgelieferten Quelltextbeschreibungen sollten Programme erzeugen, die auf jedem Amiga und sogar auf beliebigen Rechnern lauffähig sind. (Allerdings sollte man dann beim Aufruf der FlexCat-Funktionen sicherstellen, daß die Variable LocaleBase den Wert NULL hat. Lokalisierung ist aber meist nur auf dem Amiga und ab der Workbench 2.1 möglich, da erst dann die `locale.library` zur Verfügung steht.

Es ist aber prinzipiell durchaus möglich, auch unter einer früheren Workbench oder gar auf anderen Rechnern Lokalisierung anzubieten: Ein Beispiel dafür liefern die Quelltextbeschreibungsdateien '`C_c_V20.sd`' und '`C_h_V20.sd`', in denen die `locale.library` durch die `iffparse.library` ersetzt wird, falls letztere vorhanden ist, erstere dagegen nicht. Damit ist Lokalisierung schon ab der Workbench 2.0 möglich. Siehe Abschnitt 7.1 [C], Seite 9.

Zur Installation ist nichts weiter zu tun, als das eigentliche Programm an eine sinnvolle Stelle Ihres Suchpfades zu kopieren und einen geeigneten Platz für die Quelltextbeschreibungen auszuwählen. Falls Sie mit einer anderen als der englischen Sprache arbeiten wollen, müssen Sie außerdem den entsprechenden Katalog an eine geeignete Stelle kopieren. Im Falle der deutschen Sprache (die derzeit die einzige verfügbare ist) wäre die Datei '`Catalogs/Deutsch/FlexCat.catalog`'. Der einfachste Platz ist das Verzeichnis '`Locale:Catalogs/Deutsch`', möglich ist aber auch, einfach das ganze Verzeichnis '`Catalogs`' in das Directory des Programms zu kopieren. Siehe Abschnitt 7 [Benutzung], Seite 8.

3 Aufruf des Programms

FlexCat arbeitet nur vom CLI aus. Die Aufrufsyntax ist

```
FlexCat CDFILE/a,CTFILE,CATALOG/k,NEWCTFILE/k,SOURCES/m
```

Dies ist die Bedeutung der Argumente:

CDFILE ist der Name einer zu lesenden Katalogbeschreibungsdatei. Diesen anzugeben ist obligatorisch. Aus diesem Argument wird auch der Basisname bei der Quelltextbeschreibung gewonnen. Achten Sie deshalb auf Groß-/Kleinschreibung! Siehe Abschnitt 6 [Source description], Seite 6.

CTFILE ist der Name einer zu lesenden für die Erzeugung von Katalogen zu lesenden Katalogübersetzungsdatei. Außerdem kann man eine vorhandene Katalogübersetzungsdatei mit Hilfe des Argumentes NEWCTFILE auf den neuesten Stand zu bringen: Wird beides angegeben, so wird zunächst die Katalogbeschreib\-ungs- und dann die -übersetzungsdatei gelesen und anschließend eine neue Katalogübersetzungsdatei erzeugt, die dieselben Strings wie die alte und evtl. neue Strings (als Leerzeile) enthält.

CATALOG
ist der Name eines zu erzeugenden Kataloges. Dieses Argument ist nur gemeinsam mit der Verwendung CTFILE erlaubt.

NEWCTFILE
ist der Name einer neu zu erzeugenden Katalogübersetzungsdatei. Wie schon gesagt, werden die Strings aus einer evtl. durch CTFILE angegebenen bestehenden Datei übernommen. Fehlt das Argument CTFILE, so wird eine Datei erzeugt, die nur Leerzeilen als Strings enthält.

SOURCES
sind die Namen zu erzeugender Quelltextdateien sowie der dazu zu lesenden Quelltextbeschreibungsdateien. Diese Argumente müssen die Form '`source=template`' haben, wobei '`source`' der Name der zu erzeugenden Quelltextdatei und '`template`' der Name der Quelltextbeschreibungsdatei ist.

Ein Beispiel:

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

liest die Katalogbeschreibungsdatei '`Prog.cd`' und erzeugt daraus die Katalogübersetzungsdatei '`NewCatalog.ct`' sowie den Quelltext '`prog_cat.c`'. Letzterer entspricht der Quelltextbeschreibungsdatei '`C_c_V21.sd`'. Der Basisname in der Quelltextbeschreibung ist '`Prog`'.(Beachten Sie die Großschreibung!)

4 Aufbau einer Katalogbeschreibungsdatei

Eine Katalogbeschreibungsdatei enthält vier Arten von Zeilen.

Kommentarzeilen
Jede mit einem Semikolon beginnende Zeile ist eine Kommentarzeile, wird also von FlexCat ignoriert. (Eine Ausnahme sind die unten beschriebenen Stringzeilen, die sehr wohl mit einem Semikolon beginnen dürfen.)

Kommandozeilen
Mit einem '#' beginnende Zeilen enthalten ein Kommando. Mögliche Kommandos sind (Groß-/Kleinschreibung wird ignoriert):

`#language <str>`
gibt die Vorgabesprache des Programms an, d.h. die Sprache der Strings in der Katalogbeschreibungsdatei. Vorgabe ist '`#language english`'.

`#version <num>`
gibt die Versionsnummer der zu eröffnenden Kataloge an. Im Unterschied zu `Exec.OpenLibrary` muß die Nummer genau stimmen, höhere Nummern werden nicht akzeptiert. Eine Ausnahme ist es, hier die 0 als Versionsnummer anzugeben, durch die jeder Katalog akzeptiert wird. Vorgabe ist '`#version 0`'. Zu diesen Befehlen siehe auch `Locale.OpenCatalog`.

`#lengthbytes <num>`

Weist das Programm an, vor jeden String die angegebene Zahl von Bytes zu schreiben, die die Länge des Strings (ohne die lengthbytes) enthalten und ohne abschließendes NUL-Byte angeben. (Ein NUL-Byte wird in Katalogen aber trotzdem angehängt, im erzeugten Quelltext ist dies von der Quelltextbeschreibungsdatei abhängig.) '`<num>`' muß `<=` sizeof(long), d.h. `<=` 4 sein. Vorgabe ist '`#lengthbytes 0`'.

`#basename <str>`

Setzt den Basisnamen für die Quelltextbeschreibung. Der aus den Argumenten beim Aufruf des Programmnamens gewonnene Basisname (siehe Abschnitt 3 [Programmstart], Seite 2) wird überschrieben. Siehe Abschnitt 6 [Source description], Seite 6.

Beschreibungszeilen

deklarieren einen String. Sie haben die Form '`IDSTR (id/minlen/maxlen)`', wobei '`IDSTR`' ein Bezeichner ist (d.h. ein aus den Zeichen a-z,A-Z,0-9 und dem Underscore bestehender String), '`id`' eine eindeutige Nummer (die von jetzt an als ID bezeichnet wird) angibt, '`minlen`' die minimale und '`maxlen`' die maximale Länge des Strings. Die drei letztgenannten dürfen auch fehlen, das Programm wählt dann selbst einen Wert für '`id`' und erlaubt Strings beliebiger Länge. Die auf eine Beschreibungszeile folgende ist eine

Stringzeile,

d.h. sie enthält den eigentlichen String und nichts anderes. Dieser darf eine Reihe von Steuerzeichen enthalten, die alle durch einen Backslash eingeleitet werden:

'`\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)

'`\)`' Das Klammer-Zu-Zeichen. (Dies ist evtl. innerhalb einer '`%(..)`'-Sequenz nötig, siehe Abschnitt 6 [Source description], Seite 6.)

'`\\`' Der Backslash selbst.

'`\xHH`' Das durch '`HH`' gegebene Ascii-Zeichen, wobei '`HH`' Hexziffern sind.

'`\OOO`' Das durch '`OOO`' gegebene Ascii-Zeichen, wobei '`OOO`' Hexziffern sind.

Schließlich signalisiert ein einzelner Backslash am Zeilenende, daß die Zeile (und damit der String) auf der nächsten Zeile fortgesetzt wird. Es ist dadurch möglich, beliebig lange Strings zu definieren. (FlexCat ist lediglich durch das verfügbare RAM eingeschränkt.)

Ein String wird also stets durch eine Beschreibungszeile und eine unmittelbar darauffolgende Stringzeile angegeben. Ein Beispiel wäre

```
msgHello (/4/)
Hello, this is english!\n
```

In diesem Beispiel fehlt die ID, wird also vom Programm festgesetzt. (Dies ist sicher der einfachste und beste Weg.) Die 4 gibt hier an, daß der in der nächsten Zeile stehende String wenigstens 4 Zeichen enthalten soll, eine maximale Länge fehlt.

Als ausführlicheres Beispiel zum Aufbau einer Katalogbeschreibungsdatei kann die Datei '`FlexCat.cd`' dienen.

5 Aufbau einer Katalogübersetzungsdatei

Katalogübersetzungsdateien entsprechen in ihrem Aufbau ganz und gar den Katalogbeschreibungsdateien. Nur sind auf den Kommandozeilen andere Kommandos erlaubt und die Beschreibungszeilen enthalten keine Angaben über ID sowie minimale oder maximale Länge, da diese aus der Katalogbeschreibung entnommen werden. Selbstverständlich sollte jeder String aus der Katalogbeschreibung auch in der Katalogübersetzung vorkommen und es dürfen keine Strings (d.h. Stringbezeichner) auftauchen, die nicht auch in der Katalogbeschreibung definiert sind. Dies zu sichern geht am einfachsten, indem man mit FlexCat aus den evtl. geänderten Katalogbeschreibungen und den evtl. alten Katalogübersetzungen neue erzeugt. Siehe Abschnitt 3 [Programmstart], Seite 2.

Die in Katalogübersetzungsdateien erlaubten Kommandos sind:

`##version <str>`
: Gibt die Version des Kataloges in Form eines AmigaDOS-Versionsstrings an. Beispiel:

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

  Die Versionsnummer dieses Kataloges ist 8. Um ihn zu eröffnen, müssten also in der Katalogbeschreibung die Versionsnummern 0 oder 8 angegeben werden.

`##language <str>`
: Gibt die Sprache des Kataloges an. Natürlich sollte dies eine andere als die Sprache der Katalogbeschreibung sein. Die Katalogsprache und die Katalogversion **müssen** angegeben werden.

`##codeset <num>`
: Ein derzeit noch unbenutztes Argument für die Eröffnung eines Kataloges. Sollte immer 0 sein. (Dies ist auch der Vorgabewert.)

Das obige Beispiel sieht hier so aus:

```
msgHello
Hallo, dies ist deutsch!\n
```

Als weiteres Beispiel einer Katalogübersetzungsdatei kann '`Deutsch.ct`' dienen.

6 Aufbau einer Quelltextbeschreibungsdatei

Der wichtigste Teil von FlexCat ist die Quelltexterzeugung. Bis hierher bietet FlexCat nichts, was nicht auch CatComp, KitCat und Konsorten bieten würden. Der erzeugte Quelltext soll nun die Verwendung der erzeugten Kataloge möglichst einfach machen. Andererseits soll dies aber unter beliebigen Programmiersprachen und für beliebige Anforderungen gelten. Um diese scheinbaren Widersprüche aufzulösen, kennt FlexCat die Quelltextbeschreibungsdateien. Das sind Dateien, die gewissermaßen die Vorlage für den zu erzeugenden Quelltext bilden. Wie die Katalogbeschreibungs- und die -übersetzungsdateien sind sie mit einem Editor erzeug- und bearbeitbar: Das ist es, was FlexCat so flexibel macht.

FlexCat durchsucht die Quelltextbeschreibung nach gewissen Symbolen, die durch die in der Katalogbeschreibung gegebenen Werte ersetzt werden. Mögliche Symbole sind zum einen die mit einem Backslash eingeleiteten Steuerzeichen, die auch in den Strings der Katalogbeschreibung und der Katalogübersetzung erlaubt sind, zum anderen aber Steuerzeichen, die mit einem `%`-Zeichen beginnen: Für C-Programmierer ein wohlvertrautes Konzept. Mögliche Steuerzeichen sind:

'`%b`' ist der Basisname der Quelltextbeschreibungsdatei. (Für '`FlexCat.cd`' als CDFILE wäre also `FlexCat` der Basisname. Wie schon erwähnt, kommt es deshalb beim Argument CDFILE sehr wohl auf Groß-/Kleinschreibung an; siehe Abschnitt 3 [Programmstart], Seite 2)

'`%v`' ist die Versionsnummer aus der Katalogbeschreibung, nicht zu verwechseln mit dem Versionsstring aus der Katalogübersetzung.

'`%l`' ist die Sprache der Katalogbeschreibung. Bitte beachten Sie, daß hier ein String eingesetzt wird, dessen Aussehen mit dem Kommando `##stringtype` beeinflußt wird.

'`%n`' ist die Anzahl der Strings in der Katalogbeschreibung.

'`%%`' ist das Prozentzeichen selbst.

Das wesentlichste sind aber die folgenden Steuerzeichen. Sie repräsentieren auf unterschiedliche Art und Weise die Strings der Katalogbeschreibung. Zeilen die eines dieser Zeichen enthalten, werden von FlexCat für jeden Katalogstring wiederholt, da im Normalfall kaum alle Strings in eine Zeile passen würden.

'`%i`' ist der Bezeichner aus der Katalogbeschreibung.

'`%d`' ist die ID des Strings

‘`%s`’ ist der String selbst; dieser wird in einer von der Programmiersprache abhängigen Art und Weise dargestellt. Dies kann mit den Kommandos `##stringtype` und `##shortstrings` beeinflußt werden.

‘`%(...)`’ gibt an, daß der zwischen den Klammern stehende Text bei allen Strings außer dem letzten auftauchen soll. Dies ist z.B. bei Arrays nützlich, wenn unterschiedliche Arrayeinträge durch ein Komma getrennt werden sollen, nach dem letzten aber kein Komma mehr kommen soll: Dann würde man nach dem Stringeintrag eben ‘`%(,)`’ schreiben. Beachten Sie, daß der Text zwischen den Klammern nicht weiter auf ‘`%`’-Symbole untersucht wird. Backslash-Sequenzen sind allerdings weiter erlaubt.

Die Steuerzeichen ‘`%l`’ und ‘`%s`’ erzeugen Strings. Die Darstellung von Strings hängt natürlich von der Programmiersprache ab, für die Quelltext erzeugt werden soll. Deshalb können in die Quelltextbeschreibung ähnlich wie in der Katalogübersetzung Kommandos eingebaut werden. Diese müssen am Zeilenanfang stehen und jeweils eine eigene Zeile einnehmen. Die möglichen Kommandos sind:

`##shortstrings`
: gibt an, daß lange Strings über mehrere Zeilen verteilt werden dürfen. Dies ist nicht in allen Programmiersprachen ohne weiteres möglich und vor allem besonders stark von der verwendeten Programmiersprache abhängig. Deshalb werden vorgabemäßig notfalls eben sehr lange Zeilen erzeugt.

`##stringtype <art>`
: gibt die Syntax der Strings an. Mögliche Arten sind:

  None Es werden keinerlei zusätzliche Zeichen erzeugt und lediglich die Zeichen des Strings ausgegeben. Es ist keine Ausgabe von Binärzeichen (das sind die mit dem Backslash erzeugten Zeichen) möglich.

  C erzeugt Strings gemäß den Regeln der Programmiersprache C, d.h. die Strings werden links und rechts mit je einem Anführungszeichen abgegrenzt. Falls Strings über mehrere Zeilen verteilt werden, so werden die Zeilen bis auf die letzte mit einem Backslash beendet. (Der Backslash ist innerhalb von Makros nötig.) Steuerzeichen werden mit ‘`\OOO`’ ausgegeben. Siehe Abschnitt 7.1 [C], Seite 9.

  Oberon wie der Stringtyp bei C, allerdings wird kein Backslash bei Zeilentrennung erzeugt. Siehe Abschnitt 7.2 [Oberon], Seite 11.

  Assembler Strings werden mit ‘`dc.b`’ erzeugt und links und rechts mit einem einfachen Anführungsstrich abgegrenzt. Binärzeichen werden mit $XX erzeugt. Siehe Abschnitt 7.3 [Assembler], Seite 11.

Als Beispiel betrachten wir einen Auszug aus der Quelltextbeschreibungsdatei ‘`C_h.sd`’, die eine Include-Datei für die Programmiersprache C erzeugt:

```
##stringtype C
##shortstrings

#ifndef %b_CAT_H    /* Sicherstellen, daß Include-Datei    */
```

```
#define %b_CAT_H    /* nur einmal verwendet wird.      */


#ifndef EXEC_TYPES_H /*  Nötige andere Include-  */
#include <exec/types.h>  /*  Dateien einbinden.      */
#endif
#ifndef LIBRARIES_LOCALE_H
#include <libraries/locale.h>
#endif


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

/*  Definitionen der Bezeichner und ihrer ID's              */
#define %i %d /*  Diese Zeile wird für jeden Katalog-     */
/*  wiederholt.       */

#endif
```

7 Einbau des erzeugten Quelltextes in eigene Pro\-gram\-me

Wie der Quelltext benutzt wird, hängt natürlich vom erzeugten Quelltext und damit von den jeweiligen Quelltextbeschreibungen ab. Siehe Abschnitt 6 [Source description], Seite 6. Es kann hier deshalb nur auf die mit FlexCat mitgelieferten Quelltextbeschreibungsdateien eingegangen werden.

Alle diese Dateien sind so aufgebaut, daß das fertige Programm auf jeden Fall auch ohne die `locale.library` arbeitet. Allerdings muß es eine globale Variable '`LocaleBase`' (d.h. '`_LocaleBase`' für Assembler-Programmierer) geben und diese muß mit '`NULL`' oder durch einen Aufruf von `Exec.OpenLibrary` initialisiert sein. Im ersten Fall werden natürlich nur die eingebauten Strings aus der Katalogbeschreibung verwendet. Eine Ausnahme stellt die Quelltextbeschreibung '`C_c_V20.sd`' dar, die auch unter der Workbench 2.0 Lokalisierung ermöglicht, indem sie evtl. die `locale.library` durch die `iffparse.library` ersetzt. (Diese Version benötigt dann auch eine Variable '`IFFParseBase`' für die das gleiche wie für '`LocaleBase`' gilt.) Siehe Abschnitt 7.1 [C], Seite 9. Als Programmierer benötigen Sie keinerlie Kenntnisse dieser Libraries, außer Sie wollen eigene Quelltextbeschreibungen erzeugen.

Es gibt lediglich 3 Funktionen, die aufzurufen recht simpel ist:

OpenCatalog (*locale, language*)

Diese Funktion versucht, einen Katalog zu eröffnen. Das Argument `locale` ist ein Zeiger auf eine Locale-Struktur, `language` ein Zeiger auf einen String, der den Namen der gewünschten Sprache enthält. Beide Argumente werden an die Locale-Funktion OpenCatalog übergeben und sollten normalerweise immer NULL (bzw.

NIL) sein, da andernfalls die Voreinstellungen des Benutzers überschrieben werden. Näheres ist in den AutoDocs nachzulesen.

Hat der Benutzer als Vorgabesprachen etwa '`Deutsch`' und '`Francais`' eingestellt und der Basisname des Programms ist '`XXX`', so wird nacheinander nach folgenden Dateien gesucht:

```
'PROGDIR:Catalogs/Deutsch/XXX.catalog'
'LOCALE:Catalogs/Deutsch/XXX.catalog'
'PROGDIR:Catalogs/Francais/XXX.catalog'
'LOCALE:Catalogs/Francais/XXX.catalog'
```

Dabei ist '`PROGDIR:`' das aktuelle Directory des Programms. Die Reihenfolge von '`PROGDIR:`' und '`LOCALE:`' kann evtl. vertauscht werden, falls dadurch ein Requester wie '`Insert volume YYY`' unterdrückt werden kann.

OpenCatalog ist vom Typ void (für Modula2-Programmierer: Eine Prozedur), liefert also kein Ergebnis.

GetString (*ID*)

Ist der Katalog eröffnet, so erhält man mit dieser Funktion einen Zeiger auf den Katalogstring mit der angegebenen Nummer. Die ID wird in der Katalogbeschreibung definiert. Es versteht sich von selbst, daß die Strings Eigentum der `locale.library` sind und deshalb nicht verändert werden dürfen.

Ein Beispiel ist vielleicht nützlich. Im Beispiel aus der Katalogbeschreibung wird der String `msgHello` definiert. Die Quelltextbeschreibungen deklarieren nun eine Konstante '`msgHello`', der die ID repräsentiert. Damit könnte der String in C so ausgegeben werden:

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

CloseCatalog (*void*)

Mit dieser Funktion wird der Katalog (das heißt das belegte RAM) vor dem Programmende wieder freigegeben. Die Funktion kann gefahrlos zu jeder Zeit aufgerufen werden, sogar wenn OpenCatalog gar nicht aufgerufen wurde.

7.1 FlexCat-Quelltext in C-Programmen

Der C-Quelltext besteht aus zwei Teilen: Einer '`.c`'-Datei, die einfach übersetzt und mit dem Linker eingebunden wird und nicht weiter zu interessieren braucht und einer '`.h`'-Datei, die vom benutzenden Programm mit '`#include`' eingebunden wird. In ihr werden die ID's der Strings als Makro definiert.

Dabei gibt es zwei unterschiedliche Versionen: '`C_c_V21.sd`' und '`C_h_V21.sd`' sind recht simple Versionen, die einfach die entsprechenden Funktionen der `locale.library` benutzen. Dementsprechend ist mit ihnen Lokalisierung ab Workbench 2.1 möglich. Dagegen ersetzen '`C_c_V20.sd`' und '`C_h_V20.sd`' unter 2.0 evtl. die `locale.library` durch die

`iffparse.library`. Das setzt allerdings voraus, daß das Programm eine Option `Language` besitzt, die die Wahl einer Sprache ermöglicht und deren Wert dann an '`OpenCatalog`' übergeben wird. Unter 2.1 oder höher sollte diese Option nicht benutzt werden.

Es wäre prinzipiell durchaus denkbar, auch eine dritte Version zu schreiben, die Lokalisierung sogar unter 1.3 ermöglicht, aber das habe ich nicht getan, da ich 1.3 nicht mehr unterstützen möchte.

Um die FlexCat-Funktionen OpenCatalog und CloseCatalog von den gleichnamigen Locale-Funktionen zu unterscheiden und um theoretisch auch das gleichzeitige Eröffnen mehrerer Kataloge zu ermöglichen, tragen die FlexCat-Funktionen etwas geänderte Namen, nämlich '`OpenXXXCatalog`', '`CloseXXXCatalog`' und '`GetXXXString`'. Dabei ist '`XXX`' der Basisname aus der Quelltextbeschreibung. Das Konzept ist von der `GadToolsBox` übernommen und meines Erachtens bewährt. Siehe Abschnitt 6 [Source description], Seite 6.

Die Prototypen der Funktionen sind:

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

Zum Schluß noch ein Beispiel eines Programms, das FlexCat verwendet.

```
#include <stdio.h>
#include <stdlib.h>
#include <XXX.h>     /* Diese Include-Datei unbedingt */
/* einbinden!  */
#include <clib/exec_protos.h>

struct Library *LocaleBase;
/* Library auf jeden Fall selber eröffnen, auch wenn der */
/* Compiler das automatisch kann!  */

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

{
  LocaleBase = OpenLibrary("locale.library", 38);
  /*  KEIN Abbruch, falls OpenLibrary() nicht erfolg-       */
  /*  reich! (Natürlich nur, wenn Locale-Funktionen nicht   */
  /*  anderweitig benutzt werden.  */
  OpenXXXCatalog(NULL, NULL);

  ...   /* andere Funktionen     */

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

  ...   /* nochmal andere Funktionen   */

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

7.2 FlexCat-Quelltext in Oberon-Programmen

Es gibt zwei unterschiedliche Quelltextbeschreibungen: '`Oberon_V38.sd`' erzeugt Quelltext, der '`Locale.mod`' von Hartmut Goebel verwendet. Der mit '`Oberon_V39.mod`' erzeugte Quelltext benötigt dagegen das beim AmigaOberon mitgelieferte '`Locale.mod`'.

Die Prototypen der Funktionen sind:

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

Dabei ist '`XXX`' jeweils der Basisname aus der Quelltextbeschreibung. Siehe Abschnitt 6 [Source description], Seite 6.

Zum Schluß noch ein Beispiel eines Programms, das den von FlexCat erzeugten Quelltext verwendet:

```
MODULE Irgendwas;

IMPORT  x:=XXX; Dos;

BEGIN
  x.OpenCatalog(NIL, "");

  ... (* Irgendwelche anderen Funktionen. *)

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

  ... (* Nochmal irgendwelche anderen Funktionen.   *)
(* Katalog wird beim Programmende automatisch *)
(* geschlossen.                               *)
  END Irgendwas;
```

7.3 FlexCat-Quelltext in Assembler-Programmen

Der Assembler-Quelltext erzeugt Code für den Aztec-Assembler. Dieser dürfte sich aber kaum wesentlich von anderen verbreiteten Assemblern unterscheiden und es sollte ohne große Probleme möglich sein, daraus eine eigene Quelltextbeschreibung zu machen. Der Quelltext besteht aus zwei Teilen: Einer '`.asm`'-Datei, die einfach übersetzt und mit dem Linker eingebunden wird und nicht weiter zu interessieren braucht und einer '`.i`'-Datei, die vom benutzenden Programm mit '`include`' eingebunden wird. In ihr werden die ID's der Strings definiert.

Um theoretisch auch das gleichzeitige Eröffnen mehrerer Kataloge zu ermöglichen, tragen die FlexCat-Funktionen etwas geänderte Namen, nämlich '`OpenXXXCatalog`', '`CloseXXXCatalog`' und '`GetXXXString`'. Dabei ist '`XXX`' der Basisname aus der Quelltextbeschreibung. Das Konzept ist von der `GadToolsBox` übernommen und meines Erachtens bewährt. Siehe Abschnitt 6 [Source description], Seite 6.

Die Funktionen liefern wie üblich das Ergebnis in d0 und sichern die Register d2-d7 und a2-a7. OpenCatalog erwartet seine Argumente in a0 (Zeiger auf Locale-Struktur) und in a1

(Zeiger auf String mit zu verwendender Sprache). Wie schon erwähnt, sollten diese Argumente im Normalfall immer NULL sein. GetString erwartet in d0 eine ID.

Zum Schluß noch ein Beispiel eines Programms, das FlexCat verwendet.

```
include "XXX.i" /*  Enthält xref OpenXXXCatalog usw.   */

xref _LVOOpenLibrary
xref _LVOCloseLibrary
xref _AbsExecBase

dseg
LocNam: dc.b "locale.library",0
dc.l _LocaleBase,4     ; Dieser Name ist obligatorisch

cseg

main:  move.l #38,d0     ; Locale eröffnen
lea LocName,a1
move.l _AbsExecBase.a6
jsr _LVOOpenLibrary(a6)
*  KEIN Abbruch, falls OpenLibrary() nicht erfolgreich! (Natürlich nur,
*   wenn Locale-Funktionen nicht anderweitig benutzt werden.

move.l #0,a0
move.l #0,a1
jsr OpenXXXCatalog     ; Katalog eröffnen

...      ; andere Funktionen

move.l #msgHello,d0     ; Zeiger auf String holen
jsr GetXXXString
jsr PrintD0      ; und ausgeben

...      ; nochmal andere Funktionen

Ende:
jsr CloseXXXCatalog     ; Katalog schließen
move.l _LocaleBase,a1     ; Locale evtl. schließen
move.l a1,d0
beq Ende1
jsr CloseLibrary
Ende1:
rts
end
```

Weiterentwicklung des Programms

Ich beabsichtige eigentlich nicht, das Programm wesentlich weiterzuentwickeln, denke auch nicht, daß das nötig sein wird, bin aber natürlich trotzdem für jegliche Anregung, Vorschläge oder notfalls auch Kritik offen. Was ich auf jeden Fall gerne machen werde, sind andere Stringtypen, falls sich diese für andere Programmiersprachen als notwendig erweisen sollten.

Ferner wäre ich auf jeden Fall dankbar für weitere Quelltextbeschreibungen und würde diese gerne in einer späteren Version öffnetlich zugänglich machen - egal, welche Programmiersprache oder mit welchen Erweiterungen. Voraussetzung ist natürlich, daß der von diesen Quelltextbeschreibungen erzeugte Code in einem laufenden Programm erfolgreich erprobt wurde.

Ebenso dankbar wäre ich natürlich auch für neue Kataloge. Es genügt der Eintrag der entsprechenden Strings in der Datei '`NewCatalogs.ct`'. Wie das geht, sollte nach der Lektüre dieser Dokumentation hoffentlich klar sein.

Danksagungen

Danken möchte ich:

Albert Weinert
: für KitCat, den Vorgänger von FlexCat, der mir gute Dienste geleistet hat, aber irgendwann eben nicht flexibel genug war.

Reinhard Spisser und Sebastiano Vigna
: für die Amiga-Version von texinfo, mit der diese Dokumentation geschrieben ist.

Der Free Software Foundation
: für die Urversion von texinfo und für viele andere hervorragende Programme.

Matt Dillon
: für DICE und besonders für DME.

Alessandro Galassi
: für den italienischen Katalog.

Den Leuten von #AmigaGer
: für die Beantwortung vieler dummer Fragen und für viele Augenblicke erfreulich ungezügelten Schwachsinns :-), z.B. PowerStat (Kai Hoffmann), ZZA (Bernhard Möllemann), Stargazer (Petra Zeidler), stefanb (Stefan Becker), Tron (Mathias Scheler), ill (Markus Illenseer).

Commodore
: für den Amiga und für die Kickstart 2.0 :-) Macht weiter mit der Kiste, dann bin ich vielleicht auch die nächsten 8 Jahre Amiga-Benutzer!

Index

B

C

D

F

I

K

O

P

Q

S

W

Z

Inhaltsverzeichnis