FlexCat

Der flexible Kataloggenerator

Version 1.4

Jochen Wiedmann

Am Eisteich 9
72555 Metzingen (Deutschland)
Tel. 07123 / 14881
Internet: wiedmann@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. Betrachten wir als Beispiel ein Programm '`HelloLocalWorld.c`'. Das Programm wird letzten Endes so aussehen:

```
#include <stdio.h>
#include <stdlib.h>
#include <HelloLocalWorld_Cat.h>  /* Muß eingebunden werden! */

struct Library *LocaleBase;

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

Beachten Sie, daß dies dem originalen '`HelloWorld.c`' fast völlig entspricht, abgesehen davon, daß der String "Hello, world!" durch einen Funktionsaufruf ersetzt wird.

Das obige Programm verwendet eine Konstante msgHello. Ein Aufruf der Funktion `GetHelloLocalWorldString()` ersetzt diese Konstante durch den entsprechenden String. Man beginnt stets damit, diese Konstanten und Strings in einer sogenannten `Katalogbeschreibung` abzulegen. Siehe Abschnitt 4 [Catalog description], Seite 5. Unsere Katalogbeschreibung würde in einer Datei '`HelloLocalWorld.cd`' stehen und so aussehen:

```
; Kommentare sind natürlich erlaubt! Jede mit einem Semikolon
; beginnende Zeile ist eine Kommentarzeile.
;
; Die Sprache der eingebauten Strings:
#language english
;
; Die für den Aufruf von Locale/OpenCatalog() verwendete
; Versionsnummer. Dies ist anders als bei Exec/OpenLibrary():
; 0 bedeutet beliebige Version, andere Nummern müssen exakt
; stimmen!
#version 0
;
; Dies definiert einen String und die ID unter der er verwendet
; wird. Die Zahl 4 gibt an, daß der String wenigstens 4 Zeichen
; enthalten sollte.
msgHello (/4/)
Hello, world!
```

Mit FlexCat erzeugt man ais der Katalogbeschreibung zwei andere Dateien: Das Includefile '`HelloLocalWorld_Cat.h`' definiert die Konstanten, die Datei '`HelloLocalWorld_Cat.c`' enthält ein Array mit den Strings sowie die Funktion `GetString`. Wie diese genau aussieht, ist unerheblich. Insbesondere benötigt man keinerlei Kenntnisse der `locale.library`!

Allerdings könnten Sie neugierig sein, wie diese Dateien aussehen oder sogar ein anderes Aussehen wünschen. Hier liegt der Unterschied zwischen FlexCat und anderen Kataloggeneratoren: Bei FlexCat ist kein bestimmtes Format vorgeschrieben. Mit Hilfe der sogenenannten Quelltextbeschreibungen können Sie praktisch beliebige Formate vorgeben. Damit könnten z.B. auch unter AmigaDOS 2.0 Kataloge verwendet werden. Siehe Abschnitt 6 [Source description], Seite 8. Solche Katalogbeschreibungen sind bei FlexCat bereits mitgeliefert werden. Damit kann man die Quelltexte folgendermaßen erzeugen:

```
'FlexCat HelloLocalWorld.cd HelloLocalWorld_Cat.c=AutoC_c.sd'
'FlexCat HelloLocalWorld.cd HelloLocalWorld_Cat.h=AutoC_h.sd'
```

Wenn das Programm fertig ist, dann wird FlexCat erneut verwendet, um sogenannte `Katalogübersetzungen` zu erzeugen, eine für jede weitere Sprache außer der eingebauten. Siehe Abschnitt 5 [Catalog translation], Seite 7. Erzeugen wir also eine deutsche Katalogübersetzung:

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

Die fertige Datei sieht dann so aus:

```
## version
## language
## codeset 0
; Kommentare sind natürlich erlaubt! Jede mit einem Semikolon
; beginnende Zeile ist eine Kommentarzeile.
;
; Die Sprache der eingebauten Strings:
;
; Die für den Aufruf von Locale/OpenCatalog() verwendete
; Versionsnummer. Dies ist anders als bei Exec/OpenLibrary():
; 0 bedeutet beliebige Version, andere Nummern müssen exakt
; stimmen!
;
; Dies definiert einen String und die ID unter der er verwendet
; wird. Die Zahl 4 gibt an, daß der String wenigstens 4 Zeichen
; enthalten sollte.
msgHello

;Hello, world!
```

Dies sieht der Katalogbeschreibung sehr ähnlich. FlexCat übernimmt dabei die Kommentare, auch dort wo es nutzlos ist: Z.B. ist der Kommentar zur Länge der Strings hier bedeutungslos, da diese bereits in der Katalogbeschreibung angegeben werden muß. Man muß nun lediglich die Lücken füllen, d.h. die Sprache (language, hier Deutsch), die Version (einen typischen Versionsstring, '`$VER: Deutsch.catalog (11.03.94)`' wäre z.B. gut möglich) und den codeset (hier 0, siehe Locale/OpenCatalog() für Details) sowie natürlich die übersetzten Strings selber. FlexCat erleichtert dies, indem die originalen Strings jeweils als Kommentare eingefügt werden.

Schließlich werden daraus die eigentlichen Kataloge erzeugt:

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

Beachten Sie, daß man dazu weder das Programm noch die Quelltexte benötigt. Dies kann also ohne weiteres später geschehen, etwa um nachträglich weitere Sprachen zu unterstützen. Es ist üblich und durchaus erwünscht, eine Datei '`NewCatalog.ct`' mitzuliefern, die das Erstellen eigener Kataloge erlaubt.

Aber was geschieht, wenn das Programm später geändert oder erweitert wird? Dann muß nur die Katalogbeschreibung geändert werden. Mit Hilfe von FlexCat können die Katalogübersetzungen auf den neuesten Stand gebracht werden:

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

Es müssen dann lediglich noch evtl. neue Strings eingegeben werden.

2 Installation des Programms

FlexCat sollte auf jedem Amiga mit OS 2.0 laufen. Die erzeugten Programme sind auf **jedem** Amiga lauffähig (zumindest was die Verwendung der `locale.library` betrifft). Prinzipiell sind sie sogar auf anderen Rechnern lauffähig, sofern sichergestellt ist, daß die Variable *LocaleBase*

beim Aufruf der Funktion `OpenCatalog` dern Wert '`NULL`' hat. Lokalisierung ist aber natürlich nur auf dem Amiga und ab der Workbench 2.1 möglich, da erst dann die `locale.library` zur Verfügung steht. Siehe Abschnitt 7 [Benutzung], Seite 10.

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`', in der 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 11.

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. Möglicherweise wollen Sie die Umgebungsvariable *FLEXCAT_SDDIR* setzen. Siehe Abschnitt 3 [Programmstart], Seite 4.

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 wäre dies '`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 10.

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 (obligatorische) Name einer zu lesenden Katalogbeschreibung. Aus diesem Argument wird auch der Basisname bei der Quelltextbeschreibung gewonnen. Achten Sie deshalb auf Groß-/Kleinschreibung! Siehe Abschnitt 6 [Source description], Seite 8.

CTFILE ist der Name einer Katalogübersetzung, die etwa für die Erzeugung eines Katalogs zu lesen ist. Außerdem kann man eine vorhandene Katalogübersetzung mit Hilfe des Argumentes NEWCTFILE auf den neuesten Stand zu bringen: Wird beides angegeben, so wird zunächst die Katalogbeschreibung und dann die -übersetzung gelesen und anschließend eine neue Katalogübersetzung 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 CTFILE erlaubt.

NEWCTFILE
ist der Name einer neu zu erzeugenden Katalogübersetzung. 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 Quelltextbeschreibungen. Diese Argumente müssen die Form '`source=template`' haben, wobei '`source`' der Name der zu erzeugenden Quelltextdatei und '`template`' der Name der Quelltextbeschreibungsdatei ist.

Wird die angegebene Quelltextbeschreibung nicht gefunden, so sucht FlexCat nach einer Datei gleichen Namens in '`PROGDIR:lib`', d.h. im Unterverzeichnis '`lib`' des Directories, in dem sich das Programm selbst befindet. ('`PROGDIR:lib`' kann durch die Environment-Variable *FLEXCAT_SDDIR* überschrieben werden.) Beispiel: Mit

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

würde zunächst nach einer Datei '`Templates/C_c_V20.sd`' im aktuellen Directory gesucht. Würde diese nicht gefunden, und es gäbe keine Variable *FLEXCAT_SDDIR*, so würde nach einer Datei '`lib/Templates/C_c_V20.sd`' im Directory des Programms FlexCat gesucht. Gäbe es dagegen eine Variable *FLEXCAT_SDDIR* und diese hätte etwa den Wert '`Work:FlexCat`', so würde nach der Datei '`Work:FlexCat/Templates/C_c_V20.sd`' gesucht.

Für weitere Beispiele siehe Abschnitt 1 [Übersicht], Seite 1.

4 Aufbau einer Katalogbeschreibung

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 zwischen 0 und sizeof(long)=4 liegen. 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 4) wird überschrieben. Siehe Abschnitt 6 [Source description], Seite 8.

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 8.)

'`\\`' 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 Katalogübersetzung

Katalogübersetzungen entsprechen in ihrem Aufbau ganz und gar den Katalogbeschreibungen. 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 1 [Übersicht], Seite 1.

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 Quelltextbeschreibung

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 Quelltextbeschreibungen. Das sind Dateien, die gewissermaßen die Vorlage für den zu erzeugenden Quelltext bilden. Wie die Katalogbeschreibungen und -übersetzungen 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 4)

'`%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

'`%e`' gibt an, um den wievielten String (Mit 0 beginnend) es sich handelt.

‘`%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 11.

  Oberon wie der Stringtyp bei C, allerdings wird kein Backslash bei Zeilentrennung erzeugt. Siehe Abschnitt 7.3 [Oberon], Seite 14. Dieser Stringtyp wird auch für Modula-2 empfohlen.

  Assembler Strings werden mit ‘`dc.b`’ erzeugt und links und rechts mit einem einfachen Anführungsstrich abgegrenzt. Binärzeichen werden mit $XX erzeugt. Siehe Abschnitt 7.5 [Assembler], Seite 15.

  E Strings werden mit je einem ’ umgeben. Mehrzeilihe Strings werden durch ein ’+’ konkateniert. Binärzeichen win in C.

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
```

Zum Suchpfad von Quelltextbeschreibungen siehe auch Abschnitt 3 [Programmstart], Seite 4.

7 Benutzung in eigenen Programmen

Wie der Quelltext benutzt wird, hängt natürlich vom erzeugten Quelltext und damit von den jeweiligen Quelltextbeschreibungen ab. Siehe Abschnitt 6 [Source description], Seite 8. 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 11. 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 '`Français`' 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/Français/XXX.catalog'
'LOCALE:Catalogs/Français/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*)

Diese Funktion liefert 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 drei unterschiedliche Versionen: '`AutoC_c.sd`' ist die einfachste Version: Sie verwendet die Autoinitialisierungsfähigkeiten von `Dice` und `SAS/C` wodurch hier nur noch der Aufruf der Funktion `GetString` zu tun bleibt, allerdings bietet sie auch nicht alle Möglichkeiten der `locale.library`, sollte aber für sollten allerdings für 95% aller Anwendungen ausreichen sein. Zu '`AutoC_c.sd`' gehört auch noch `AutoC_h.sd`, das die Includefiles erzeugt. Ein Beispiel eines so geschriebenen C-Programms findet man in Abschnitt 1 [Übersicht], Seite 1.

Die Quelltextbeschreibungen '`C_c_V12.sd`' und '`C_h.sd`' braucht man dagegen, wenn man

1. einen anderen Compiler als SAS/C oder Dice benutzt,
2. alle Möglichkeiten der Funktion `locale.library/OpenCatalogA` benutzen möchte oder
3. mehrere Kataloge in einem Programm benutzen möchte.

Um letzteres zu ermöglichen, gibt es hier die Funktionen `OpenXXXCatalog`, `GetXXXString` und `CloseXXXCatalog`, wobei '`XXX`' der Basisname aus der Quelltextbeschreibung ist. Siehe Abschnitt 6 [Source description], Seite 8. Dies sind die Prototypen:

```
VOID OpenXXXCatalog(struct Locale *loc, char *language);
STRPTR GetXXXString(APTR);
VOID CloseXXXCatalog(VOID);
```

Hier muß man die Initialisierung und Terminierung, d.h. die Eröffnung der `locale.library` und den Aufruf der Funktionen `OpenXXXCatalog` und `CloseXXXCatalog` selbst übernehmen. Ein auf diesen Quelltextbeschreibungen basierendes Programm sieht so aus:

```
#include <stdio.h>
#include <stdlib.h>
#include <HelloLocalWorld_Cat.h> /* Muß eingebunden werden! */
#include <clib/exec_protos.h>

struct Library *LocaleBase;

void main(int argc, char *argv[])
{
  /*  Eröffne locale.library; Kein Abbruch, falls nicht
vorhanden! (Dann werden einfach die eingebauten Strings
verwendet.
Aus diesem Grund muß die locale.library auch selbst
eröffnet werden, selbst wenn der Compiler das automatisch
kann.
    */
    LocaleBase = OpenLibrary("locale.library", 38);
    OpenHelloLocalWorldCatalog(NULL, NULL);

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

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

Die dritte Version besteht schließlich aus '`C_c_V20.sd`' und '`C_h.sd`'. Sie ist funktional identisch mit der zweiten Version, allerdings wird unter 2.0 versucht, die `locale.library` durch die

`iffparse.library` zu ersetzen und so wenigstens die Kataloge doch zu verwenden. Das Programm hat dann für gewöhnlich eine Option '`LANGUAGE`', die die Wahl einer Sprache ermöglicht und deren Wert dann an `OpenXXXCatalog` übergeben wird.[1]

7.2 FlexCat-Quelltext in C++-Programmen

Unter C++ ist alles dank einer geeigneten Klasse extrem einfach: Durch Konstruktoren und Destruktoren wird das meiste automatisch erledigt. Diese Klasse ist in den Dateien '`C++_CatalogF.cc`' und '`C++_CatalogF.h`' implementiert, die in '`CatalogF.cc`' und '`CatalogF.h`' umbenannt und übersetzt werden sollten. Ferner sollten mit Hilfe der Quelltextbeschreibungen '`C++_cc.sd`' und '`C++_h.sd`' zwei weitere Dateien erzeugt werden: Erstere enthält die Strings, letztere wird mit `#include` im eigentlichen Programm eingebunden und enthält die nötigen Deklarationen.

Abschließend ein Beispiel eines C++-Programms:

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

struct LocaleBase *LocaleBase = 0;

int main()
{ //  Die Library muß hier eröffnet werden, auch wenn der Compiler
  //  das automatisch kann. Kein Aussteigen, falls die
  //  Library nicht eröffnet werden kann: Dann werden einfach die
  //  eingebauten Strings verwendet.
  LocaleBase = (struct LocaleBase *) OpenLibrary("locale.library", 38);

  const CatalogF cat(0, 0, HelloLocalWorld_ARGS);

  cout >> cat.GetString(msgHelloLocalWorld);

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

Für den gcc ist eine Modifikation der '`libauto.a`' verfügbar, die sogar die Eröffnung der Locale.library überflüssig macht.

[1] Es wäre übrigens prinzipiell auch denkbar, eine auch unter 1.3 laufende Version zu schreiben, die den Katalog nötigenfalls "von Hand" liest, aber ich möchte 1.3 nicht mehr unterstützen und habe das deshalb nicht getan.

7.3 FlexCat-Quelltext in Oberon-Programmen

Es gibt unterschiedliche Quelltextbeschreibungen: '`AmigaOberon.sd`' ist für die aktuelle Version des AmigaOberon-Compilers, '`Oberon_V39.sd`' für ältere Versionen dieses Compilers gedacht. '`Oberon_V38.sd`' verwendet das von '`Locale.mod`' von Hartmut Goebel. '`Oberon-A.sd`' ist natürlich für den gleichnamigen Compiler.

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 8.

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

```
MODULE HelloLocalWorld;

IMPORT  x:=HelloLocalWorld_Cat; Dos;

BEGIN
  x.OpenCatalog(NIL, "");

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

  (* Katalog wird beim Programmende automatisch *)
  (* geschlossen.                               *)
END Irgendwas;
```

7.4 Flexcat-Quelltext in Modula-2-Programmen

Wie Oberon unterstützt Modula-2 ein Modulkonzept, die Funktionsnamen sind also dieselben. Da Modula je ein Definitions- und Implementationsmodul benötigt, müssen diese durch die Quelltextbeschreibungen '`Modula2Def.sd`' bzw. '`Modula2Mod.sd`' erzeugt werden, die an den M2Amiga-Compiler angepasst sind. Außerdem wird die Definitionsdatei '`OptLocaleL.def`' benötigt, die mit dem Update auf M2Amiga v4.3 ausgeliefert wird.

Die Prototypen der Funktionen sind:

```
  PROCEDURE OpenCatalog(loc : ld.LocalePtr;
language : ARRAY OF CHAR);
  PROCEDURE CloseCatalog();
  PROCEDURE GetString(num : LONGINT) : ld.StrPtr;
```

Zum Schlu"s noch ein Beispiel eines Programms, das den von FlexCat erzeugten Quelltext verwendet:

```
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-Quelltext in Assembler-Programmen

Es gibt Quelltextbeschreibungen für die Assembler von Aztec-C bzw. SAS/C. Beide sind im wesentlichen identisch, vor allem der SAS-Assembler dürfte sich kaum von verbreiteten Assemblern unterscheiden. Es sollte also kein großes Problem sein, daraus eine eigene Quelltextbeschreibung zu machen. Der Quelltext besteht aus zwei Teilen: Einer '`.asm`'- bzw. '`.a`'-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 8.

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 a0 einen Zeiger. Auf was er zeigt braucht ebenfalls nicht zu interessieren.

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

```
*   HelloLocalWorld.asm

include "HelloLocalWorld_Cat.i"
/*  Enthält xref OpenHelloLocalWorldCatalog 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.

sub.l a0,a0
sub.l a1,a1
jsr OpenHelloLocalWorldCatalog  ; Katalog eröffnen

lea.l msgHello,a0     ; Zeiger auf String holen
jsr GetHelloLocalWorldString
jsr PrintD0      ; und ausgeben

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

7.6 FlexCat-Quelltext in E-Programmen

Seit der Version 3.0 erlaubt E das Aufteilen von Programmen in separate Module. Im Folgenden wird nur die Quelltextbeschreibung '`E30b.sd`' beschrieben, die ab E3.0b funktionieren sollte. (Version 3.0a enthält signifikante Fehler, für frühere Versionen gibt es '`E21b.sd`', das allerdings die Einbindung des von FlexCat erzeugten Quelltextes in den eigentlichen Quelltext von Hand erfordert.)

'`E30b.sd`' erzeugt ein Modul '`Locale`', das eine Variable namens `cat` vom Typ '`catalog_XXX`' bereit stellt. (Wobei '`XXX`' der Basisname aus der Quelltextbeschreibung ist, siehe Abschnitt 6 [Source description], Seite 8) Eine Datei '`HelloLocalWorld.e`' könnte so aussehen:

```
    MODULE '*Locale'
-> Einbindung des Moduls

    DEF cat : PTR TO catalog_HelloLocalWorld
-> Diese Variable enthält die ganzen Katalogstrings und stellt
-> einige Methoden bereit. Sie muß in jedem Modul deklariert
-> werden, das den Katalog verwendet, allerdings nur im
-> Hauptmodul initialisiert werden.

    PROC main()

localebase := OpenLibrary('locale.library', 0)
    -> Locale.library eröffnen; Kein Abbruch, falls
    -> nicht vorhanden! (In diesem Fall werden die eingebauten
    -> Strings verwendet.
```

```
NEW cat.create()
cat.open()
    -> Diese Initialisierung wird (wie schon erwähnt) nur im
    -> Hauptmodul durchgeführt.

WriteF('\s\n', cat.msg_Hello_world.getstr())
    -> cat.msg_Hello_world ist einer der in cat enthaltenen
    -> Strings. Dieser stellt eine Methode getstr() bereit,
    -> die den Katalog liest und einen Zeiger auf den zu
    -> verwendenden String liefert.

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

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 sowie für die Oberon-Quelltextbeschreibungen.

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.

Lionel Vintenat
für die E-Quelltextbeschreibung und ihre Dokumentation, den französichen Katalog und Fehlermeldungen.

Antonio Joaquín Gomez Gonzalez (u0868551@oboe.etsiig.uniovi.es)
für die C++-Quelltextbeschreibung, den spanischen Katalog, die Übersetzung der Dokumentation in Spanisch sowie für den guten Tip über die schnellere GetString-Routine.

Olaf Peters (op@hb2.maus.de)
für die Modula-2-Quelltextbeschreibung.

Russ Steffen (steffen@uwstout.edu)
für die Anregung der FLEXCAT_SDDIR-Variablen.

Lauri Aalto (kilroy@tolsun.oulu.fi)
für die finnischen Quelltextübersetzungen.

Marcin Orlowski (carlos@felix.univ.szczecin.pl)
für die polnischen Quelltextübersetzungen

Den Leuten von #AmigaGer
für die Beantwortung vieler dummer Fragen und für viele Augenblicke erfreulich ungezügelten Schwachsinns :-), z.B. 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
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

D

E

F

I

K

M

O

P

Q

S

W

Z

Table of Contents