Docs/FlexCat_svenska

COLLABORATORS			
	TITLE : Docs/FlexCat_svenska		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 9, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Docs/FlexCat_svenska

1.1 Docs/FlexCat_svenska.guide

```
                FlexCat V2.0 - Dokumentation
****************************

   Den här filen beskriver FlexCat version 2.0, ett program som genererar
kataloger, samt källkoden för att använda dem. FlexCat liknar CatComp
och KitCat, men kan generera vilken källkod du vill.  Detta sker genom så
kallade källkodsbeskrivningar, som är mallar för den genererade koden.
De kan redigeras, och kan sålunda anpassas till olika programspråk samt
individuella behov.


Generellt:


                Anvsvarsfråntagande
                             Upphovsrätt, (INGEN) garanti

                Översikt
                                        Vad är FlexCat?

                Installation
                                    Hur får jag det att fungera?

Använda FlexCat:


                Programstart
                                     Använda FlexCat i ett skal

                Inställningar
                                   Ändra FlexCats normala beteende

                Katalogbeskrivning
                                Katalogbeskrivningar (.cd-filer)

                Katalogöversättning
                             Katalogöversättningar (.ct-filer)
```

1.2 FlexCat_svenska.guide/Disclaimer

Upphovsrätt och anndra rättsliga saker
****

Jochen Wiedmann
Am Eisteich 9
72555 Metzingen
Tyskland

Sedan version 1.8 utvecklas programmet av

Marcin Orlowski
ul. Radomska 38
71-002 Szczecin
Polen

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

1.3 FlexCat_svenska.guide/Survey

```
                Översikt
********
```

I och med OS 2.1 erbjuder Amigan ett trevligt system för att använda program på olika språk: locale.library. (Detta kallas att lokalisera programmet; det är vad namnet står för.)

Idén är enkel: Du väljer ett språk - engelska för det mesta - och skriver ditt program på samma sätt som du gjorde utan lokalisering, förutom att konstanta strängar ersätts av vissa speciella funktionsanrop. Ett annat funktionsanrop gör det möjligt för användaren att välja ett annat språk när programmet körs. (Det senare funktionsanropet läser in en extern fil, den så kallade katalogen, och gör att den ovan nämnda funktionen läser strängarna från katalogen istället för att använda de fördefinierade strängarna.)

Dessa kataloger är oberoende av programmet. Allt du behöver för att lägga till ett annat språk är att skapa en ny katalogfil, och detta kan göras när som helst, utan att programmet ändras.

Men det är några saker till som programmeraren behöver göra: Han behöver skapa katalogen, de fördefinierade strängarna, och lite källkod för att använda strängarna (de funktioner som nämns ovan). FlexCat har konstruerats för att göra detta på ett enkelt och näst intill automatiskt sätt, utan att för den skull förlora flexibilitet när det gäller skapandet av källkoden. Ett exempel för att förtydliga:

Antag att vi vill skriva HelloLocalWorld.c. Vårt slutliga program kommer att se ut så här:

```
#include <stdio.h>
#include <stdlib.h>
/* Du måste inkludera den här filen! */
#include <HelloLocalWorld_Cat.h>

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

Observera att detta är mycket likt den ursprungliga HelloWorld.c, förutom att vi har ersatt strängen "Hello, world!" med konstanten msgHello.

Dessa konstanter och de relaterade strängarna definieras i en så kallad katalogbeskrivning (se
katalogbeskrivning
). Du börjar alltid med att skapa en sådan fil, kallad HelloLocalWord.cd, som kan se ut så här:

```
; Kommentarer kan naturligtvis användas! Varje rad som
; börjar med ett semikolon antas vara en kommentar.
```

```
;
; Språket för de inbyggda strängarna:
#language english
;
; Katalogens version. Den använd vid anropet av
; Locale/OpenCatalog(). Detta skiljer sig från
; Exec/OpenLibrary(): 0 betyer vilken katalog som helst,
; medans andra nummer måste stämma exakt!
#version 0
;
; Detta definierar en sträng och det ID som används för att
; ange den. Talet 4 här säger att strängen inte får vara
; kortare än 4 tecken.
msgHello (/4/)
Hello, world!
```

Genom att använda FlexCat skapar du två filer utifrån katalogbeskrivningen: Inkluderingsfilen HelloLocaleWorld_Cat.h, som definierar konstanterna, samt filen HelloLocalWorld_Cat.c, som innehåller en vektor med strängar samt några initieringsfunktioner. Du behöver inte veta vad funktionerna gör; använd dem bara. Du behöver framför allt inte veta något om locale.library!

Du kanske är intresserad av hur dessa filer ser ut, eller du kanske till och med vill modifiera dem. Det är här skillanden mellan FlexCat och andra kataloggeneratorer märks: Med FlexCat är du inte bunden till vissa inbyggda format på dessa filer. Istället använder FlexCat externa mallfiler, så kallade källkodsbeskrivningar. Detta gör det möjligt att till exmpel använda kataloger med AmigaOS 2.0 (se
källkodsbeskrivning
). Om du använder källkodsbeskrivningar från FlexCat-distributionen kan du skapa källkodsfiler med följande kommandon:

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

När ditt program är klart, kan du använda FlexCat för att skapa så kallade katalogöversättningar; en översättning för varje språk du vill stödja (förutom det inbyggda språket). Se
katalogöversättning
. Låt oss skapa en tysk katalogöversättning för vårt exempelprogram:

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

Den här filen skulle nu se ut så här:

```
## version
## language
## codeset 0
; Kommentarer kan naturligtvis användas! Varje rad som
; börjar med ett semikolon antas vara en kommentar.
;
; Språket för de inbyggda strängarna:
;
; Katalogens version. Den använd vid anropet av
```

```
     ; Locale/OpenCatalog(). Detta skiljer sig från
     ; Exec/OpenLibrary(): 0 betyer vilken katalog som helst,
     ; medans andra nummer måste stämma exakt!
     ;
     ; Detta definierar en sträng och det ID som används för att
     ; ange den. Talet 4 här säger att strängen inte får vara
     ; kortare än 4 tecken.
     msgHello

     ;Hello, world!
```

Som du ser är de mycket lika katalogbeskrivningar. FlexCat inkluderar kommentarerna i från katalogbeskrivningen, även där de är meningslösa: Observera kommentaren om stränglängden, som inte borde synas här, eftersom den informationen bara finns i katalogbeskrivningen.

Allt du behöver göra nu är att fylla i informationen för versionen (i stil med $VER: HelloLocalWorld.catalog 1.1 (25.8.97)), språket på katalogöversättningen (Deutsch för tyska i det här fallet), koduppsättningen (som alltid bör vara 0 för tillfället; se Locale/OpenCatalog() för mer information) och naturligtvis strängen. FlexCat inkluderar den ursprungliga strängen som en kommentar, så att du alltid vet som skall fyllas i. Slutligen skapar du katalogen med ett kommando som:

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

Observera att du inte behöver själva programmet, eller källkodsfilerna som skapades med FlexCat, när du skapar katalogerna! Du kan skapa nya kataloger när som helst. Det är vanligt att i distributioner skicka med en fil som FlexCat.ct, så att användare kan göra sina egna översättningar.

Men vad händer om du ändrar programmet senare? Redigera bara katalogbeskrivningen och använd FlexCat för att uppdatera katalogöversättningarna:

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

Allt du behöver göra nu är att översätta eventuella nya strängar.

1.4 FlexCat_svenska.guide/Installation

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

FlexCat är skrivet i ren ANSI-C (förutom lokaliseringen), så det bör fungera på vilken Amiga som helst, och förhoppningsvis även på andra datorer efter en omkompilering (lokaliseringen kommenteras bort i det fallet). Detta gäller också för de skapade programmen: FlexCat skrevs med sig själv. All distribuerad källkodsbeskrivning bör skapa program som kan köras på vilken Amiga som helst, och även vilken dator som helst (naturligtvis måste du se till att variabeln LocaleBase har värdet NULL i

det senare fallet). Lokalisering är dock bara möjligt med Workbench 2.1
och högre, eftersom locale.library inte finns tillgängligt i tidigare
versioner av operativsystemet.

   Det är inte omöjligt att lokalisera ett program utan locale.library:
Källkodsbeskrivningarna C_c_V20.sd och C_h_V20.sd använder
iffparse.library (plus lite egen kod) istället för locale.library, om
locale.library inte är tillgängligt. Detta gör det möjligt att
lokalisera program som kör under Workbench 2.0. Se
                C
                .

   Att installera FlexCat är enkelt: Kopiera bara programmet till en låda
i din kommandosökväg, och välj ett ställe för de källkodsbeskrivningar du behöver
(filer som heter något i stil med xx_yy.sd, där xx är programspråket).
Du vill förmodligen sätta miljövariabln FLEXCAT.PREFS eller
FLEXCAT_SDDIR. Se
                programstart
                .

   Om du vill använda FlexCat i ett annat språk än engelska behöver du
kopiera motsvarande katalogfil. T.ex., för tyska, kopiera filen
Catalogs/Deutsch/FlexCat.catalog till Locale:Catalogs/Deutsch/ eller till
ProgDir:Catalogs/Deutsch/, där ProgDir: är den låda där FlexCat finns.
Se
                att använda FlexCat-källkod
                .

1.5 FlexCat_svenska.guide/Program start

                Använda FlexCat i ett skal
****

   FlexCat är ett skalbaserat program, och kan inte användas från
Workbench. Syntaxen är:

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

   Notera att FlexCat inte använder de normala rutinerna för att tolka
argumenten (för portabilitetens skull). Detta gör framför allt att de
enda nyckelod du kan (och måste) ange är CATALOG och NEWCTFILE (de som
är av typen "/K"), de andra kan inte anges, då de skulle tolkas som
argument i sig. Detta kommer att ändras, förmodligen i nästa version.

   I och med version 1.9 har FlexCat ett enkelt system för inställningar,
som låter dig ändra FlexCats normala beteende. Se
                inställningar
                .

   Vad argumenten används till:

CDFILE
     är namnet på den katalogbeskrivning som skall läsas. Denna måste alltid anges. Observera att basnamnet för källkodsbeskrivningen skapas från detta argument, vilket gör det lägeskänsligt. Se

          källkodsbeskrivning
          .

CTFILE
     är namnet på den katalogöversättning som skall läsas. Denna behövs när du skapar en katalog, eller när du uppdaterar en gammal katalogöversättning via NEWCTFILE-argumentet: FlexCat läser den gamla översättningen samt katalogbeskrivningen och skapar en ny katalogöversättning innehållande de gamla strängarna, och kanske några tomma rader för de nya strängarna.

CATALOG
     är namnet på den katalogfil som skall skapas. Detta argument kräver att CTFILE också anges.

NEWCTFILE
     är namnet på den katalogöversättning som skall skapas. FlexCat läser strängarna från CTFILE, om CTFILE angavs; strängar som saknas i katalogöversättningen ersätts med tomma rader. (Den nya katalogöversättningen kommer bara att innehålla tomma rader som strängar om CTFILE inte anges.)

SOURCES
     är namnet på de källkoder som skall skapas. Dessa anges i formatet källkod=mall, där källkod är filen som ska skapas, och mall är namnet på den källkodsbeskrivning som ska läsas.

     Om källkodsbeskrivningen inte kunde hittas, försöker FlexCat öppna en fil med samma namn i lådan ProgDir:lib. (Lådan lib i den låda som FlexCat finns i.) Du kan ändra denna förvalda låda genom att sätta miljövariabeln FLECCAT_SDDIR. Exempel:

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

     letar efter filen Templates/C_c_V20.sd i den aktuella lådan. Om denna fil inte kunde hittas, och miljövariablen FLEXCAT_SDDIR inte var definierad, letar FlexCat efter ProgDir:lib/Templates/C_c_V20.sd. Om FLEXCAT_SDDIR var definierad och innehöll till exempel Work:FlexCat, letar FlexCat efter Work:FlexCat/Templates/C_c_V20.sd.

WARNCTGAPS
     Normalt sett varnar FlexCat inte om symboler saknas i katalogöversättningen. Med denna flagga kan du få FlexCat att visa sådana varningar.

NOOPTIM
     Om båda strängarna (orginalet i #?.cd, och översättningen i #?.ct) är likadana, så antar FlexCat normalt sett att strängen inte behöver skrivas till katalogen, eftersom strängen då kan tas från de inbyggda strängarna istället. Men om du av någon anledning vill att även dessa strängar skall skrivas (eller: om du vill att alla strängar skall skrivas), använd NOOPTIM.

FILL

Denna finess är mycket användbar för översättarna. När du arbetar på en översättning kan en del strängar vara tomma, men du vill prova de översättningar du redan har gjort. Olyckligtvis så skriver då alla kataloggeneratorer tomma strängar, vilket skulle göra att t.ex. knappar blir tomma.

En flagga för att förbjuda tomma strängar är inte en bra lösning, eftersom du då inte kan ha sådana om det behövs. Dessutom, ett illa skrivet program kanske kräver att alla strängar finns i katalogen (även de tomma), t.ex. för att det inte finns några inbyggda strängar. Genom att använda FILL-flaggan kan du tvinga FlexCat att skriva orginalsträngarna (från #?.cd-filen) varje gång den hittar en tom översättning, eller en översättning som inte finns.

OBS: Detta bör bara användas för att prova översättningen. Färdiga kataloger bör alltid skapas utan någon FILL-flagga!

FLUSH

Denna flagga är användbar när du översätter och provar din översättning. Eftersom AmigaOS inte direkt tar bort kataloger (och bibliotek, teckensnitt, drivrutiner, etc.) från minnet när de inte längre används, kan du behöva tvinga systemet att ta bort dem (t.ex. genom att använda C:Avail FLUSH) varje gång du vill att den nyskapade katalogen skall läsas från disk (istället för att kopian i minnet används). Om du anger den här flaggan när katalogen skapas, kommer FlexCat att ta bort alla oanvända saker från minnet.

OBS: FLUSH fungerar bara när du skapar en katalog. Annars kommer den att ignoreras.

Example:

```
FlexCat Test.cd Test.ct CATALOG Test.catalog FLUSH
```

NOBEEP

I och med version 1.9 blänker FlexCat skärmen för att uppmärksamma dig om problem som har påträffats. Detta kan vara mycket användbart när du använver FlexCat från en miljö utan en normal utmatning (t.ex. om du startar ett skript från Opus eller ett annat program). Du kanske inte tycker om dessa blänkningar (fast FlexCat är smart nog att bara blänka en gång, även om du får 20 varingar). I så fall kan du använda NOBEEP-flaggan för att stänga av blänkandet.

NOLANGTOLOWER

Normalt konverterar FlexCat argumentet till #language till gemener genom att använda en funktion i utility.library. Denna funktion använder locale.library om det är installerat, men tydligen finns det felaktiga översättningstabeller för några språk (till exempel tjeckiska), vilket kan leda till en felaktig konvertering. Med den här flaggan kan du undvika problemet. Du bör dock kontakta författaren av språkfilerna så att dessa fel rättas, eftersom andra program kan drabbas av samma problem. Kom ihåg att bara använda gemener för #language om du använder den här flaggan (och använd den bara om det är absolut nödvändigt).

```
NOBUFFEREDIO
     FlexCat använder buffrad IO. Detta kan göra FlexCat snabbare, men
     det är inte säkert. Du kan då använda den här flaggan för att
     använda obuffrad IO.  Använd den bara om det är nödvändigt.

MODIFIED
     Den här flaggan gör att FlexCat bara skapar katalogen när någon av
     källfilerna, #?.cd och #?.ct, har ändrats sedan katalogen senast
     kompilerades. När katalogen är nyare än källfilerna avslutar
     FlexCat utan att göra något. Den här flaggan är användbar när du
     vill skapa en kommandofil för att kompilera flera kataloger på en
     gång (till exempel för översättningen av stora program), och inte
     vill slösa tid på att kompilera om kataloger som inte har ändrats.
     (Detta är alltså ett enklare alternativ till att använda ett
     make-program.)

QUIET
     Säger åt FlexCat att hålla tyst om det inte är absolut
     nödvändigt. Det betyder att du inte kommer att se några varningar.
     Fel kommer att rapporteras som vanligt.

   För fler exempel på kommandorader, se
                översikten
                .
```

1.6 FlexCat_svenska.guide/Preferences

```
                Ändra FlexCats normala beteende
*******************************

   I och med version 1.9 har FlexCat ett enkelt system för inställningar.
Genom att använda miljövariabeln FLEXCAT.PREFS kan du ändra programmets
normala beteende.

   Variabeln FLEXCAT.PREFS tolkas med dos.library:s ReadArgs()-funktion,
och sålunda skall alla argument anges på en rad, där argumenten separeras
med mellanslag. Mallen för inställningarna ser ut som följer:

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

   För mer information om dessa argument, Se
                programstart
                .

   En kommentar angående SDDIR: När källkoden skapas letar FlexCat
först i den aktuella lådan, sedan i lådorna som anges i inställningarna.
Om detta fortfarande inte lyckas, läses FLEXCAT_SDDIR, och slutligen lådan
ProgDir:lib/. Genom att använda både inställningsvariabeln och
FLEXCAT_SDDIR kan du alltså använda två egna lådor för
källkodsbeskrivningar samtidigt.
```

1.7 FlexCat_svenska.guide/Description

```
                Katalogbeskrivningar
********************

   En katalogbeskrivning innehåller fyra olika sorters rader.

Kommentarsrader
     En rad om börjar med ett semikolon antas vara en kommentarsrad, och
     ignoreras därmed. (Strängrader (se nedan) är ett undantag. De kan
     börja med ett semikolon.)

Kommandorader
     En rad som börjar med ett "#" (med samma undantag som ovan) antas vara
     kommandorader. Följande kommandon kan användas:

    #language <sträng>
          anger programmets förvalda språk; det språk som används i
          katalogbeskrivningen. Förvalt är #language english.

    #version <num>
          anger versionsnumret för katalogen som skall öppnas. Observera
          att detta nummer måste stämma exakt, och inte vara samma eller
          högre, som för 'Exec/OpenLibrary'. Undantaget är 0, vilket
          godtar vilken katalogversion som helst. Förvalt är #version 0.

          Se Locale/OpenCatalog för mer information om katalogspråket och
          versionen.

    #lengthbytes <num>
          Säger åt FlexCat att placera det givna antalet tecken före en
          sträng, innehållande strängens längd. Längden är det antal
          tecken strängen tar upp, förutom längdtecknena och ett
          avslutande nolltecken. (Katalogfiler, och därmed
          katalogsträngar kommer att ha ett avslutande nolltecken. Detta är
          inte alltid sant för de inbyggda strängarna, beroende på hur
          källkodsbeskrivningen ser ut.) <num> måste vara mellan 0 och
          sizeof( long ) = 4. Förvalt är #lengthbytes 0.

    #basename <sträng>
          Anger basnamnet för programmet. Se
                källkodsbeskrivning
                . Denna
          sträng används istället för basnamnet från
          kommandoradsargumentet CDFILE.  Se
                programstart
                .

     Kommandon är lägeskänsliga.

Beskrivningsrader
     deklarerar en sträng. De ser ut som IDSTR (id/minlen/maxlen), där
     IDSTR är en identifierare (en sträng bestående av tecknena a-z, A-Z
     0-9, samt _); id är ett unikt nummer (härmed kallat ID) och minlen,
     samt maxlen är strängens minimala respektive maximala längd. De tre
     sista talen behöver inte anges (men tecknena (//) måste anges!), i
```

vilket fall FlexCat väljer ett nummer, och sätter inte några begränsningar på strängens längd. Det är bäst att inte ange några ID-nummer om du inte absolut behöver det. Dessa rader följs av

Strängrader
som innehåller strängen, och inget annat. Dessa strängaar kan innehålla vissa kontrolltecken, som börjar med ett bakstreck:

\b
Backsteg (ASCII 8)

\c
Kontrollsekvensstart (ASCII 155)

\e
Escape (ASCII 27)

\f
Sidmatning (ASCII 12)

\g
Blänk (ASCII 7)

\n
Radmatning (ASCII 10)

\r
Radretur (ASCII 13)

\t
Tab (ASCII 9)

\v
Vertikal tab (ASCII 11)

\)
Avslutande parantes, som kan behövas som en del av sekvensen (..), se
källkodsbeskrivning
.

\
Bakstreck

\xHH
Tecknet som anges av ASCII-koden HH, där HH är hexadecimala siffror.

\OOO
Tecknet som anges av ASCII-koden OOO, där OOO är oktala siffror.

Slutligen, ett ensamt bakstreck på slutet av raden anger att strängen fortsätter på nästa rad. Detta gör det möjligt att ange strängar av godtycklig längd; FlexCat gör inte några antanganden gällande strängens längd.

En sträng anges alltså av en beskrivningsrad, samt den följande

```
strängraden. Ett exempel:

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

ID-numret saknas här, så FlexCat väljer ett lämpligt nummer. Numret 4
säger åt FlexCat att den följande strängen inte får ha färre än fyra
tecken, och att den i övrigt kan ha godtycklig längd. Se filen FlexCat.cd
för ytterligare exempel.
```

1.8 FlexCat_svenska.guide/Translation

```
                Katalogöversättningar
**********************

   Katalogöversättningar påminner mycket om katalogbeskrivningar,
förutom att den innehåller andra kommandon, samt ingen information om
sträng-ID och längd (dessa tas från katalogöversättningen). Alla
strängar från katalogbeskrivningen måste finnas med i översättningen
(FlexCat undviker att skriva strängar som är identiska med orginalet till
katalogen), och inga fler identifierare får finnas med. Man kan enkelt
försäkra sig om detta genom att använda FlexCat för att skapa nya
katalogöversättningar. Se
                översikt
                .

   De kommandon som tillåts i en katalogöversättning är:

##version <sträng>
     Anger katalogens version i form av en versionssträng i
     AmigaDOS-format.  Exempel:

          ##version $VER: FlexCat.catalog 8.2 (25.8.97)

     Versionen på denna katalog är 8. Sålunda måste
     katalogbeskrivningens versionsnummer vara 0 eller 8.

     Du kan ersätta datumet (här 27.09.93) med det speciella nyckelordet
     $TODAY. När katalogen skapas, kommer $TODAY att ersättas med det
     aktuella datumet (observera att bara den första förekomsten av $TODAY
     i $VER-strängen kommer att behandlas). Om du vill att dina kataloger
     alltid skall vara aktuella, ange alltså:

          $VER: FlexCat.catalog 3.1 ($TODAY)

##rcsid $Date: <datum> <tid> $ $Revision: <rev> $
$Id: <namn> <datum> <tid> carlos Exp carlos $
     kan användas tillsammans med ett revisionskontrollsystem istället
     för ##version (allt skall givetvis vara på en och samma rad). <datum>
     är datumet på formen åå/mm/dd, <tid> är tiden (ignoreras), <rev>
     är revisionen och <namn> är namnet som används i versionsträngen.

##name <namn>
```

```
     finns med för kompatibilitet med CatComp. Det ersätter
     <namn>-argumentet i ##rcsid-kommandot.

##language <sträng>
     Katalogens språk. Naturligtvis skall detta vara ett annat språk än
     det som används i katalogbeskrivningen. Kommandona ##language och
     ##version måste finnas med i en katalogöversättning.

##codeset <nummer>
     Används för tillfället inte, och måste vara 0. Detta är det
     förvalda värdet.

## chunk <ID> <sträng>
     Lägger till IFF-blocket <ID> till katalogen, innehållande den
     angivna <sträng>. Används oftast för att lägga till en kommentar
     till en katalog. Exempel:

          ## chunk AUTH Tysk översättning av Jochen Wiedmann

   Enligt ovanstående ser alltså vårt exempel ut så här i
katalogöversättningen:

         msgHello
         Hallo, dies ist deutsch!\n

Se Deutsch.ct för ytterligare ett exempel på en katalogöversättning.
```

1.9 FlexCat_svenska.guide/Source

```
                Källkodsbeskrivningar
**********************

   Detta är den speciella delen av FlexCat. Hittils har det inte varit
något som inte CatComp, KitCat och andra program också kan göra. Den
skapade källkoden gör det enkelt att använda kataloger, utan att för
den skull förlora flexibilitet. Vilket programspråk som helst bör vara
möjligt att använda, och alla behov borde kunna uppfyllas. Det låter
kanske som en motsägelse, men FlexCats lösning på detta problem är
källkodsbeskrivningarna, som innehåller en mall för hur källkoden skall
skall se ut. Dessa kan redigeras på samma sätt som katalogbeskrivningar
och katalogöversättningar. Alltså kan FlexCat generera i princip
godtycklig kod.

   Källkodsbeskrivningarna genomsöks efter vissa speciella symboler, som
ersätts med vissa värden. Möjliga symboler är bakstrecksstecknena enligt
ovan, och ytterligare sekvenser som börjar med ett %-tecken (detta är
välbekant för C-programmerare).

%b
     är basnamnet på katalogbeskrivningen. Se
                programstart
                .
```

%v
     är versionsnumret på katalogbeskrivningen. Blanda inte ihop detta med
     katalogversionen från katalogöversättningen.

%l
     är katalogbeskrivningens språk. Observera att detta matas in som en
     sträng. Se %s nedan.

%n
     är antalet strängar i katalogbeskrivningen.

%%
     är tecknet %.

   De viktigaste är följande sekvenser. De representerar
katalogsträngarna på olika sätt. Rader innehållande en eller flera av
följande symboler upprepas en gång för varje sträng.

%i
     är identifieraren från katalogbeskrivningen.

%nd
%nx
%nc
     är ID-numret för strängen i decimal, hexadecimal respektive oktal
     notation. Numret n säger åt FlexCat hur många tecken ID-numret
     skall använda (strängen fylls med noll till vänster till den
     önskade längden). Om du inte anger n kommer ID-numret bara att
     använda det antal tecken som behövs.

%e
     är numret på den aktuella strängen. Räknaren börjar från 0.

%s
     är strängen själv. Hur denna skall matas in beror på
     programspråket som används, och kan kontrolleras med kommandona
     ##stringtype och ##shortstrings.

%na
     är strängens ID. Skillnaden mellan %na och t.ex. %nx är att %na
     genererar strängens ID separerad till enstaka tecken:

          %2a i källkodsbeskrivningen resulterar i \x00\x20

     Om du inte anger n, kommer ID-numret att bli fyra tecken.

%nt
     är strängens längd. Observera att det värdet alltid är jämnt.

%z
     bör användas tillsammans med %nt. Eftersom %nt alltid resulterar i
     ett jämnt värde, kan en beskrivningsrad som:

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

orsaka problem, speciellt när man tolkar en sådan tabell, eftersom %2t kan vara jämn, medans strängens längd är udda! När tabellen tolkas kan du läsa eller hoppa över ett tecken för mycket (jag antar konsekvenserna är kända). För att undvika detta introduerades %z. FlexCat ersätter den med så många tecken (\x00) som strängen saknar för att längden skall bli jämn. Om strängen är tre tecken lång, ger %nt 4, och %z lägger till en \x00.

%(...)
matar in texten mellan paranteserna för varje sträng utom den sista. Detta kan behövas i vektorer, om posterna i vektorn skall separeras med komma, men den sista posten inte får följas av ett komma. Du kan använda (,) i så fall. Observera att inom paranteser ersätts inte några %-sekvenser. Bakstrecksekvenser ersätts dock fortfarande.

Kontrollsekvenserna %l och %s skapar strängar. Hur strängarna ser ut beror på programspråket i fråga. Därför kan man ha kommandorader i källkodsbeskrivningen liknande dem i t.ex. katalogöversättningen. Dessa kommandon måste börja på första positionen på en rad, och det kan bara finnas ett kommando per rad. Möjliga kommandon är:

##shortstrings
gör så att långa strängar delas upp på flera raader. Detta är inte alltid möjligt, eller kanske inte implementerad FlexCat. Sålunda är det normala att skapa en sträng, som kanske är mycket lång.

##stringtype <typ>
Talar om för FlexCat hur strängarna skall se ut. Möjliga typer är:

None
Inga extra tecken; en ren kopia av strängen infogas, och inget annat. Binära tecken (bakstegsekvenser) är inte möjliga.

C
skapar strängar i C-format. Strängarna föregås och följs av tecknet ". Strängar delas med "\ på slutet av raden, och " på början av nästa rad. (Baksteget behövs i makron.) Binära tecken infogas som \OOO. Se
C
.

Oberon
ser ut som strängtypen C, utom det avslutande baksteget på slutet av rader. Se
Oberon
. Denna strängtyp rekommenderas även för Modula-2.

Assembler
Strängas skapas med dc.b. Läsbara ASCII-tecken föregås och följs av tecknet ’; binära tecken infogas som $XX. Se
assembler
.

E

```
          Strängar föregås och följs av tecknet '. Ett +-tecken binder
          samman strängar som har delats upp på flera rader. Binära
          tecken infogas som i C.

   Låt oss se på ett utdrag från filen C_h.sd, som skapar en
inkluderingsfil för programspråket C:

     ##stringtype C
     ##shortstrings

     /* Se till att vi bara inkluderas en gång. */
     #ifndef %b_CAT_H
     #define %b_CAT_H

     /* Läs andra inkluderingsfiler. */
     #include <exec/types.h>
     #include <libraries/locale.h>

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

     /* Definitioner för identifierarna samt deras ID-nummer. */
     /* Den här raden kommer att upprepas för varje sträng.   */
     #define %i %d

     #endif

   För den sökväg som används för källkodsbeskrivningar, se

                programstart
                .
```

1.10 FlexCat_svenska.guide/Using source

```
                Använda källkoden i egna program
*******************************

   Hur du använder källkoden beror naturligtvis på vilken källkod som
skapas, och därmed på källkodsbeskrivningen som används. Vad vi talar
om här är de källkodsbeskrivningar som följer med FlexCat. Se

                källkodbeskrivning
                .

   Alla källkodsbeskrivningar borde göra det möjligt att använda
programmen utan locale.library. En global variabel kallad LocaleBase
(_LocaleBase för assembler) måste dock finnas, och initieras med NULL
eller med ett anrop till 'Exec/OpenLibrary'. Lokalisering är inte möjlig
om LocaleBase är NULL, såvida inte källkodsbeskrivningen C_c_V20.sd
används. Denna beskrivning möjliggör lokalisering under AmigaOS 2.0 genom
att ersätta locale.library med iffparse.library (plus lite egen kod). (En
```

variabel kallad IFFParseBase måste finnas för att detta skall fungera,
och den behöver initieras på liknande sätt som LocaleBase.) Se
C
.
Programmeraren behöver inte veta hur dessa bibliotek används, utom när
han skapar egna källkodsbeskrivningar.

Det finns tre funktioner, och de är ganska enkla att använda.

- : OpenCatalog(locale, language)
Denna funktion öppnar eventuellt en katalog. Argumentet locale är en pekare till en Locale-struktur, och language är en sträng som innehåller namnet på det språk för vilket en katalog skall öppnas. I de flesta fall är dessa båda NULL eller NIL, eftersom användarens val inte används annars. Se 'Locle/OpenCatalog' för mer information.

Icke objektorienderade språk (C, assembler, Modula) anropar normalt sett funktionen OpenXXXCatalog, där XXX är basnamnet på applikationen: Detta låter dig använda flera olika kataloger i ett och samma program.

Om användaren har Deutsch och Français som de förvalda språken, och programmets basnamn är XXX, så letar programmet efter följande filer:

```
ProgDir:Catalogs/Deutsch/XXX.catalog
Locale:Catalogs/Deutsch/XXX.catalog
ProgDir:Catalogs/Français/XXX.catalog
Locale:Catalogs/Français/XXX.catalog
```

där ProgDir: är den låda där programmet finns. (Ordningen mellan ProgDir: och Locale: kan ändras för att undvika en dialogruta i stil med Mata in disketten YYY.)

OpenCatalog är av typen void (en procedur för Pascal-programmerare), och returnerar därmed inte något värde.

- : GetString(ID)
Returnerar en pekare till den sträng som motsvaras av det givna ID:n i katalogbeskrivningen. Dessa strängar ägs av locale.library, och får inte modifieras.

Ett exempel kan vara användbart. Ta strängen från exemplet med katalogbeskrivningen, som kallades msgHello. Källkodsbeskrivningen deklarerar konstanten msgHello, som representerar strängen. Denna sträng kan skrivas ut i C med:

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

- : CloseCatalog()
Den här funktionen friar katalogen (alltså det allokerade minnet) innan programmet avslutas. Du kan anropa den här funktionen när som helst, även före OpenCatalog anropas.

C

```
          FlexCat-källkod i C-program.

C++
       FlexCat-källkod i C++-program.

Oberon
    FlexCat-källkod i Oberon-program.

Modula-2
  FlexCat-källkod i Modula-2-program.

Assembler
 FlexCat-källkod i Assembler-program.

E
          FlexCat-källkod i E-program.

Appendix
  Stöd för flera kataloger.
```

1.11 FlexCat_svenska.guide/C

```
                FlexCat-källkod i C-program
============================
```

C-källkoden består av två delar: En .c-fil som skall kompileras och användas som den är, samt en inkluderingsfil som ska inkluderas i alla källkodsfiler som använder katalogsträngar. Denna inkluderingsfil definierar alla strängars ID:n som makron.

De C-kompilatorer jag kan (SAS/C, DICE och GCC) kan automatiskt öppna bibliotek och initiera katalogerna. Du behöver alltså inte anropa funktionerna OpenCatalog och CloseCatalog; kompilatorn gör det åt dig. Dessutom anropar OpenCatalog GetString-funktionen för alla katalogsträngar. Detta gör att du helt enkelt kan skriva msgHello istället för GetString(msgHello).

Om du definierar symbolen LOCALIZE_V20 (-D LOCALIZE_V20 för DICE och GCC; DEF LOCALIZE_V20 för SAS/C), kommer du att få ett program som kan använda kataloger under OS 2.0: locale.library ersätts med iffparse.library i så fall. Ditt program behöver då även ett argument som LANGUAGE, så att användaren kan ange vilket språk som skall användas. Funktionen InitXXXCatalog (där XXX är basnamnet på ditt program) behöver anropas, med argumentet från LANGUAGE som parameter. Denna parameter ignoreras naturligtvis om du har locale.library. (Det vore möjligt att göra liknande saker under OS 1.3, men jag vill inte stödja denna föråldrade version längre.)

Du förlorar lite funktionalitet med denna källkodsbeskrivning: Till exempel kan du inte skicka med en Locale-struktur till OpenCatalog. De allra flesta programmen kommer dock inte att sakna något; andra behöver modifiera källkodsbeskrivningen.

För ett exempel på ett program som använder dessa
källkodsbeskrivningen, se
översikten
.

OBS:

I och med version 1.9 innehåller arkivet källkodsbeskrivningen CatComp_h.sd, som kan användas av program som använder mer än en katalog. Se den filen för hur du kan uppdatera de andra källkodsbeskrivningarna.

Det finns också en ny källkodsbeskrivning av Magnus Holmgren <cmh@lls.se>. Filerna Cat2h_c.sd och Cat2h_h.sd innehåller källkodsbeskrivningar som genererar kod liknande den som genereras av Cat2h av Nico François (och även av Cat2Inc av Magnus Holmgren ;). Den använder ett lite annorlunda sätt för att hantera strängar, som är kompakt och snabbt.

Istället för att lagra alla strängar i en vektor, och söka igenom denna varje gång (som CatComp normalt sett gör; man kan undvika detta dock), lagras strängens ID i de två första tecknena av strängen. GetString()-funktionen, som då alltså tar en sträng som argument, läser de två första tecknena till ett långord, och därmed är strängens ID och den inbyggda strängen kända.

I och med version 1.9 kan FlexCat generera den här sortens kod, via kommandot %a. De inkluderade filerna använder %2a, och använder alltså bara två byte för ID-nummer per string (som Cat2h gör). Detta torde räcka för de flesta program. Om du ändrar längden, kom ihåg att GetString()-funktionen också behöver ändras.

Den genererade inkluderingsfilen definierar alla strängar, och källkodsfilen innehåller kod för att öppna och stänga katalogen (med automatisk initiering för SAS/C och DICE), och en lämplig GetString()-funktion. En snabb titt på den genererade koden borde vara nog för att se alla detaljer, tycker jag.

Koden stödjer för tillfället inte flera kataloger, ej eller ändring av versionsnummer eller inbyggt språk. Enkelt att lägga till dock, om behovet skulle uppstå.

1.12 FlexCat_svenska.guide/C++

FlexCat-källkod i C++-program.
==========

Att använda FlexCat-källkod i C++-program är mycket enkelt: Nästan allt görs av en speciell klass, som implementeras i filerna C++_CatalogF.cc och C++_CatalogF.h. All du behöver göra är att döpa om dessa filer till CatalogF.cc och CatalogF.h, kompilera dem, och skapa ytterligare två filer genom att använda källkodsbeskrivningarna C++_cc.sd och C++_h.sd. Den första skapar filen med strängarna (som

naturligtvis också måste kompileras), och den andra filen kommer att inkluderas i ditt program. Ett C++-program som använder FlexCat-källkod kan se ut så här:

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

     struct LocaleBase *LocaleBase = 0;

     int main()
     { // Du måste öppna bibliteket här, även om din kompilator
        // stödjer automatisk öppning: Det kommer vanligtvis att
        // avsluta programmet om locale.library inte kunde öppnas.
        // Det är inte vad vi vill här, eftersom vi då kommer att
        // använda de inbyggda strängarna istället.
        LocaleBase = ( struct LocaleBase * )
            OpenLibrary( "locale.library", 38 );

        const CatalogF cat( 0, 0, HelloLocalWorld_ARGS );

        cout >> cat.GetString( msgHelloLocalWorld );

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

   En modifiering av GCCs libauto.a finns tillgänglig, som låter dig ta bort raderna som gäller variabeln LocaleBase.

1.13 FlexCat_svenska.guide/Oberon

```
                FlexCat-källkod i Oberon-program
================================
```

   Det finns några olika källkodsbeskrivningar: AmigaOberon.sd är gjord för den aktuella versionen av kompilatorn AmigaOberon. Oberon_V39.sd är för äldre versioner, och Oberon_V38.sd använder filen Locale.mod från Hartmut Goebel. Oberon-A.sd är, naturligtvis, för Oberon-A.

   Funktionsprototyperna är:

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

där XXX är basnamnet från källkodsbeskrivningen. Se

```
                källkodsbeskrivning
                .
```

```
   Till slut ett exempel på hur man använder FlexCat-källkoden:

     MODULE HelloLocalWorld;

     IMPORT  x:=HelloLocalWorld_Cat; Dos;

     BEGIN
         x.OpenCatalog(NIL, "");

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

         (* Katalogen kommer automatiskt att stängas *)
         (* när programmet avslutas.                 *)
     END Anything;
```

1.14 FlexCat_svenska.guide/Modula-2

```
                FlexCat-källkod i Modula-2-program.
==================================

   Modula-2 har ett modulkoncept liknande det i Oberon. Detta betyder att
funktionsnamnen är alltid samma. Till skillnad från Oberon behöver dock
Modula-2 en implementations- och en deklarationsmodul. Därför behöver du
skapa två filer, genom att använda källkodsbeskrivningarna Modula2Def.sd
och Modula2Mod.sd. Dessa är anpassade för kompilatorn M2Amiga. Observera
att du även behöver filen OptLocalL.def från version 4.3 av M2Amiga.

   Funktionsprototyperna är:

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

där XXX är basnamnet från källkodsbeskrivningen.  Se

                källkodsbeskrivning
                .

   Till slut ett exempel på hur man använder FlexCat-källkoden:

     MODULE HelloLocalWorld;

     IMPORT hl: HelloLocalWorldLocale,
            io: InOut;

     BEGIN
         hl.OpenCatalog(NIL, "");

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

         hl.CloseCatalog;
```

```
    END HelloLocalWorld.
```

1.15 FlexCat_svenska.guide/Assembler

```
                FlexCat-källkod i assemblerprogram
==================================
```

Assemblerkällkoden har gjorts för att användas med Aztec-assemblern. Denna bör dock inte skilja sig så mycket från andra assemblatorer, och du bör kunna göra egna källkodsbeskrivningar. Källkoden består av två delar: En .asm-fil, som assembleras och länkas, samt en .i-fil, som definierar strängarnas ID, och måste inkluderas av de moduler som använder dem.

Funktionsnamnen har ändrats lite för att tillåta användandet av olika kataloger i en fil: Dessa är OpenXXXCatalog, CloseXXXCatalog och GetXXXString, där XXX är basnamnet från källkodsbeskrivningen. Konceptet har kopierats från GadToolsBox, och har visat sig fungera bra. Se

```
                källkodsbeskrivning
                .
```

Som vanligt returneras funktionsresultatet i d0, och funktionerna ändrar inte på registrena d2-d7 och a2-a7. OpenCatalog förväntar sig sina argument i a0 (en pekare till Locale-strukturen) och a1 (en pekare till språksträngen), vilka för det mesta är NULL. GetString förväntar sig en pekare i a0. Du behöver inte bry dig om vad den pekar på.

Till slut ett exempel på hur man använder FlexCat-källkoden:

* HelloLocalWorld.asm

```
        include "XXX.i" ; Du måste inkludera denna. Den innehåller
                        ; "xref OpenHelloLocalWorldCatalog", ...

        xref    _LVOOpenLibrary
        xref    _LVOCloseLibrary
        xref    _AbsExecBase

        dseg

LocNam: dc.b    "locale.library",0
        dc.l    _LocaleBase,4       ; Måste finnas med detta namn.

        cseg

main:   move.l  #38,d0              ; Öppna locale.library
        lea     LocName,a1
        move.l  _AbsExecBase.a6
        jsr     _LVOOpenLibrary(a6)
```

* Avsluta INTE, om OpenLibrary misslyckas

```
            sub.l   a0,a0               ; Öppna katalogen
            sub.l   a1,a1
            jsr     OpenHelloLocalWorldCatalog

            lea.l   msgHello,a0         ; Hämta en pekare till strängen
            jsr     GetHelloLocalWorldString
            jsr     PrintD0             ; och skriv ut strängen

    Ende:
            jsr     CloseHelloLocalWorldCatalog ; Stäng katalogen
            move.l  _LocaleBase,a1      ; Stäng locale.library
            move.l  a1,d0               ; Denna test måste göras under 1.3
            beq     Ende1

            jsr     CloseLibrary
    Ende1:
            rts
            end
```

1.16 FlexCat_svenska.guide/E

```
                FlexCat-källkod i E-program
============================
```

I och med version 3.0 kan ett E-program delas upp i moduler. Den följande beskrivningen beskriver hur E30b.sd används, som fungerar med E version 3.0b eller högre. (Version 3.0a hade allvarliga fel; för tidigare versioner kan man använda E21b.sd, vilket kräver att man manuellt infogar den genererade koden i den egna koden.)

E30b.sd skapar en modul som heter Locale, som innehåller en variabel, cat, av typen catalog_XXX, där XXX är basnamnet från källkodsbeskrivningen. Se

```
                källkodsbeskrivning
                .  Filen HelloLocalWorld.e
```

kan se ut något sådant här:

```
    MODULE '*Locale'
        -> Använd den här modulen.

    DEF cat : PTR TO catalog_HelloLocalWorld
        -> Den här variablen innehåller alla strängarna i
        -> katalogen, samt några metoder. Du måste deklarera
        -> den i alla moduler som använder lokalisering, men
        -> initiera den bara i huvudmodulen.

    PROC main()
        localebase := OpenLibrary('locale.library', 0)
            -> Öppna locale.library; avsluta inte om den
            -> inte kunde öppnas: Vi kommer att använda de
            -> inbyggda strängarna i så fall.

        NEW cat.create()
```

```
        cat.open()
            -> Som redan har nämnts, detta skall bara göras i
            -> huvudmodulen.

        WriteF('\s\n', cat.msg_Hello_world.getstr())
            -> cat.msg_Hello_world är en av de strängar som
            -> finns i cat. Den här strängen deklarerar en
            -> metod, getstr(), som läser katalogen och
            -> returnerar en pekare till den lokaliserade
            -> strängen.

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

1.17 FlexCat_svenska.guide/Appendix

```
Stöd för flera kataloger
========================

   De flesta källkodsbeskrivningar som är tillgängliga för tillfället
kan inte användas i program som öppnar mer än en katalog. I senare
distributioner kommer detta säkerligen att ändras, och uppdaterade
källkodsbeskrivningar kommer att vara en del av de distributionerna.

   För tillfället följer det med ett exempel av en sådan
källkodsbeskrivning.  Se CatComp_h.sd för att se hur beskrivningen kan
justeras för att undvika att få flera symboler med samma namn och
liknande. Med några få ord: Använd %b som prefix, suffix eller infix i
varje namn som är en viktig del av källkoden (och som dessutom är
globalt synliga). Om din tabell med strängar heter String, ersätt den med
%b_Strings, och du kommer inte längre att få flera symboler med samma
namn.

   CatComp_h.sd producerar källkod som liknar den som CatComp genererar,
och kan användas av de som vill använda FlexCat, men inte vill nämnvärt
ändra sina program.
```

1.18 FlexCat_svenska.guide/Future

```
                Vidare utveckling av FlexCat
*****************************

   Även om FlexCat verkar vara så gott som klart, har jag några poster
kvar på min "att göra" lista. Och jag är naturligtvis öppen för
förslag, tips eller kritik. Speciellt kan jag lägga till nya strängtyper,
eftersom detta kan göras med mycket små ändringar.

   Jag vore mycket tacksam om någon skickade mig nya
```

källkodsbeskrivningar och om jag kunde inkludera dem i framtida
distributioner. Vilket programspråk som helst, och vilka utökningar som
helst, under förutsättning att de har visat sig fungera bra genom tester
i riktiga program. Se
                support
                 för kontaktadresser.

1.19 FlexCat_svenska.guide/Support

FlexCat-support
****

   För programuppdateringar, besök hemsidan för FlexCat vid:
     http://amiga.com.pl/flexcat/

   Om du har några förslag eller felrapporter, skicka e-post till:

     carlos@amiga.com.pl

   eller via vanlig post:

     Marcin Orlowski
     ul. Radomska 38
     71-002 Szczecin
     Poland

1.20 FlexCat_svenska.guide/Credits

Tack till
****

   Jochen Wiedmanns tack går till:

Albert Weinert
     för KitCat, föregångaren till FlexCat, som har gjort värdefulla
     saker för mig, men till slut inte var flexibel nog, samt för
     källkodsbeskrivningen för Oberon.

Reinhard Spisser und Sebastiano Vigna
     för Amiga-versionen av TexInfo, som användes för den här
     dokumentationen.

The Free Software Foundation
     för den ursprungliga versionen av TexInfo, och många andra utmärkta
     program.

Matt Dillon
     för DICE och speciellt DME.

Alessandro Galassi
     för den italienska översättningen.

Lionel Vintenat
     för källkodsbeskrivningen för E samt dess dokumentation, den franska
     översättningen, samt för felrapporter.

Antonio Joaquín Gomez Gonzalez (u0868551@oboe.etsiig.uniovi.es)
     för källkodsbeskrivningen för C++, den spanska översättningen av
     både program och manual, och för det mycket bra förslaget för att
     snabba upp GetString-funktionen.

Olaf Peters (op@hb2.maus.de)
     för källkodsbeskrivningen för Modula-2.

Russ Steffen (steffen@uwstout.edu)
     för att ha föreslagit FLEXCAT_SDDIR-variabeln.

Lauri Aalto (kilroy@tolsun.oulu.fi)
     för den finska översättningen.

Marcin Orlowski (carlos@inet.com.pl)
     för den polska översättningen, och för underhållet av det polska
     locale-paketet.

Udo Schuermann (walrus@wam.umd.edu)
     för att ha föreslagit WARNCTGAPS-flaggan och ##chunk-kommandot.

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

Personerna på #AmigaGer
     för att ha svarat på många dumma frågor, och för mycket roligt.
     Till exempel stefanb (Stefan Becker), PowerStat (Kai Hoffmann), ill
     (Markus Illenseer), Quarvon (Jürgen Lang), ZZA (Bernhard Möllemann),
     Tron (Mathias Scheler), mungo (Ignatios Souvlatzis), jow (Jürgen
     Weinelt) och Stargazer (Petra Zeidler).

Commodore
     för Amiga och Kickstart 2.0. Fortsätt utveckla Amigan, och jag
     kommer att vara en Amiga-användare för de nästa åtta åren också.
     ;-)

   Marcins tack går till:

Jochen Wiedmann
     för att ha skapat FlexCat.

Magnus Holmgren <cmh@lls.se>
     för källkodsbeskrivningen Cat2h.

Medlemmar i Amiga Translators' Organization <http://ato.vapor.com/ato/>
     för att ha skapat nya översättningar, och uppdaterat existerande
     översättningar:

    Serbiska
          av Ljubomir Jankovic <lurch@afrodita.rcub.bg.ac.yu>.

```
    Tjeckiska
          av Vit Sindlar <xsindl00@stud.fee.vutbr.cz>.

    Svenska
          av Magnus Holmgren <cmh@lls.se> och Hjalmar Wikholm
          <hjalle@canit.se>.

    Finska
          uppdaterad av Mika Lundell <c71829@uwasa.fi>.

    Italienska
          omarbetad av Luca Nora <ln546991@silab.dsi.unimi.it> och Giovanni
          Addabbo <gaddabbo@imar.net>.
```

1.21 FlexCat_svenska.guide/History

```
Utveklingshistoria
******************

   Utvecklingshistorien för FlexCat finns i filen FlexCat.history, som är
en del av distributionsarkivet.
```

1.22 FlexCat_svenska.guide/Index

```
                Index
*****


                 Ändringar
                                            History

                 Översikt
                                             Survey

                 .cd
                                                   Description

                 .ct
                                                   Translation

                 .sd
                                                   Source

                 Adress
                                                Disclaimer

                 AmigaOberon
                                           Oberon
```

Använda källkoden
Using source

ASCII-kod
Description

Assembler
Assembler

AutoC_c.sd
C

AutoC_h.sd
C

AztecAs_asm.sd
Assembler

AztecAs_i.sd
Assembler

Bidrag
Future

C
C

C++
C++

C++_CatalogF.cc
C++

C++_CatalogF.h
C++

C++_cc.sd
C++

C++_h.sd
C++

Cat2h_c.sd
C

Cat2h_h.sd
C

CATALOG
Program start

Catcomp_h.sd
C

CatComp_h.sd
Appendix

NOOPTIM
Program start

Oberon
Oberon

Oberon-A
Oberon

Oberon_V38.sd
Oberon

Oberon_V39.sd
Oberon

QUIET
Program start

Skal
Program start

SOURCES
Program start

Support
Support

Systemkrav.
Installation

Tack till
Credits

Tillstånd
Disclaimer

Upphovsrätt
Disclaimer

WARNCTGAPS
Program start

Workbench
Program start