catalog

COLLABORATORS			
	TITLE : catalog		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 9, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

catalog

1.1 catalog.doc

```
--overview--

def()

end()

get()

open()
```

1.2 catalog.m/--overview--

```
 PURPOSE
To act as the base class for localized ID/string pairs.

 OVERVIEW
The locale library contains a mechanism for isolating textual
strings in program code (such as "OK", "Please select file" and
other strings for the benefit of the user) and allowing for them
to be changed externally from the program, while still holding
'default' copies of the strings in the program core. This creates
the opportunity to have 'localized' strings.

The default mechanism for this is with two 'catalog' files, Bla.cd
and Bla.ct, where the .cd file is the catalog of 'descriptors' and
the .ct file is a catalog of translations for a specific language.

The .cd file contains your original text strings, and an uppercase
identifier for each of them. Example:

MSG_OPEN_FILE (//)
Open file...

The identifier is translated into a numerical ID, and using the
```

'const' / 'define' feature of most programming languages, the ID name is associated with that unique numerical value.

When you wish to use a string in your program, you would get it from a function taking the ID and returning the string.

```
filereq('Open file...')
BECOMES
filereq(get(MSG_OPEN_FILE))
```

What the catalog_obj does is represent the catalog as an object, thereby allowing you easy use of multiple catalogs, and using even single catalogs very easily. The general mechanism for actually using catalogs is to subclass the bare catalog object and set up default preset values - catalog name, builtin strings, language of builtin strings, and possibly version.

EXAMPLE

See the included example files showing a localised 'helloworld'.

A FlexCat '.sd' file, included and used by the example, can be used for automatic generation of compiled catalog modules from .cd files.

1.3 catalog.m/def

NAME

catalog_obj.def() -- Set up the default strings block.

SYNOPSIS

```
def(block:PTR TO LONG)
```

FUNCTION

Set the catalog's block of default strings. This is best kept as a static list with constants and static strings in it - the catalog neither copies the strings nor the block into its own memory, it uses the block and strings direct from whatever you pass it.

INPUTS

block - a list containing ID/string pairs, see the example below.

EXAMPLE

```
CONST MSG_HELLO=100, MSG_BYE=101
catalog.def([
  MSG_HELLO, 'Hello',
  MSG_BYE',  'Bye'
])
```

NOTE

The block _must_ be terminated with a long containing NIL. Amiga E does this automatically when you use the list operator, but not when you use the builtin-assembler 'LONG' directive, or magic a list up yourself.

In other words, just use this as shown and you'll be fine.

```
 SEE ALSO

              get()
```

1.4 catalog.m/end

```
 NAME
catalog_obj.end() -- Destructor.

 SYNOPSIS
end()

 FUNCTION
Frees resources used by an instance of the catalog_obj class.
```

1.5 catalog.m/get

```
 NAME
catalog_obj.get() -- Get a localized message string.

 SYNOPSIS
string := get(id)

 FUNCTION
Attempts  to  retrieve  a  string from the opened catalog with the
requested  ID value. If not in the catalog, it will look for it in
the  block of default strings. If not there either, it will return
NIL.

 INPUTS
id - the ID value associated with the required string.

 RESULT
string - the  requested  string,  possibly  translated,  or NIL if
         there is no such string available.
```

1.6 catalog.m/open

```
                  NAME
catalog_obj.open() -- Constructor.

 SYNOPSIS
open(catalog, language, locale:PTR TO locale)

 FUNCTION
Creates an instance of the catalog class. Opens the locale library
and appropriate translations of the catalog as neccesary.
```

```
As  no  minimum  level  of  allocations  are  necessary to operate
correctly, this constructor throws no exceptions.

Apart from calling this constructor, you should also call
              def()
               to
set up default strings before starting to call
              get()
              .

 INPUTS
catalog  - The  name  of  the catalog containing your strings, for
           example 'helloworld.catalog'. Must not be NIL.

language - A  string  representing the name of the language of the
           default strings you supply - for example, 'français' or
           'deutsch'.  If  the language of the defaults strings is
           'english', you can pass NIL instead.

locale -   If  you  are  using  a particular locale structure from
           OpenLocale(),  you  may  pass  this  in  to  affect the
           opening and parsing of the catalog. If you just want to
           use  the default locale (user's preference) then simply
           pass NIL.

 SEE ALSO
locale.library/OpenCatalog(),
              def()
```