

catalog

| |
|----------------------|
| COLLABORATORS |
|----------------------|

| | | | |
|---------------|---------------------------|--------------|------------------|
| | <i>TITLE :</i> catalog | | |
| <i>ACTION</i> | <i>NAME</i> | <i>DATE</i> | <i>SIGNATURE</i> |
| WRITTEN BY | | May 28, 2025 | |

| |
|-------------------------|
| REVISION HISTORY |
|-------------------------|

| NUMBER | DATE | DESCRIPTION | NAME |
|--------|------|-------------|------|
| | | | |

Contents

| | | |
|----------|----------------------------------|----------|
| 1 | catalog | 1 |
| 1.1 | catalog.doc | 1 |
| 1.2 | catalog.m/--overview-- | 1 |
| 1.3 | catalog.m/def | 2 |
| 1.4 | catalog.m/end | 2 |
| 1.5 | catalog.m/get | 3 |
| 1.6 | catalog.m/open | 3 |

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 versions of the strings in the program's core. This creates the opportunity to have 'localized' strings.

The default mechanism for this is with two files - Bla.cd and Bla.ct, where the .cd file is the 'catalog' of 'descriptors' - 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 `helloworld.e`, `helloworld_test.e`, `helloworld.cd` and `helloworld.ct` for an example of usage.

A FlexCat 'source descriptor' is included for automatic generation of catalog modules from `.cd` files.

1.3 catalog.m/def

NAME

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

SYNOPSIS

`def(block)`

`def(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 or 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'
])
```

SEE ALSO

`get()`

1.4 catalog.m/end

NAME

`catalog_obj.end()` -- Destructor.

SYNOPSIS

`end()`

FUNCTION

Closes catalog and other things associated with the object.
Use the END statement rather than calling end() explicitly.

1.5 catalog.m/get

NAME

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

SYNOPSIS

string := get(id)

PTR TO CHAR := get(LONG)

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 - a LONG containing the ID value for 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(catalogname, language, locale)

open(PTR TO CHAR, PTR TO CHAR, PTR TO locale)

FUNCTION

Creates a new instance of the catalog object. Opens the locale library and appropriate translations of the catalog as necessary.

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

catalogname - 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', then 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()`