

recent

Rüdiger Hanke

COLLABORATORS

	<i>TITLE :</i> recent		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Rüdiger Hanke	August 25, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	recent	1
1.1	Welcome to recent.library	1
1.2	User Intro	1
1.3	Installation	2
1.4	Troubleshooting	2
1.5	Future	3
1.6	Contact	3
1.7	Using recent.library in your program	4
1.8	Library Functions	5
1.9	AllocRecentHandle	5
1.10	FreeRecentHandle	6
1.11	CloseRecentHandle	6
1.12	AddRecentFile	6
1.13	AddRecentGroup	7
1.14	GetRecentInfo	8
1.15	SaveRecentList	9
1.16	SetRecentSize	9
1.17	GetRecentMenu	10
1.18	RecentInfo Structure	10
1.19	Style Guide	11
1.20	Developer Help	12
1.21	Tutorial	12
1.22	Group ID List	16

Chapter 1

recent

1.1 Welcome to recent.library

recent.library 0.9 and this guide are copyright (c)1999 by Rüdiger Hanke

recent.library User Guide

- Introduction
- Installation
- Troubleshooting
- Future
- Contact

recent.library Developer Guide

- Using recent.library in your program
- Developer Help
- Library Functions
- Style Guide
- A Quick Tutorial

recent.library was written using the DevPac 2 assembler and GoldEd 6.2.0

1.2 User Intro

Welcome to recent.library

recent.library is an AmigaDOS library that helps application programmers keeping track of the most recently opened files. Programs using recent.library can offer you a menu option to quickly select a file you opened with the program lately. While this can be achieved without a library, programs using recent.library offer the user more than just access to recent files from the menu bar:

In the works is a program for Workbench/DOPus that allows the user accessing his recently used files without having to load the application, for example to quickly copy them to disk or start them with another program, e.g. viewing a picture with a fast image viewer instead of

loading the entire image processing application. This tool will allow to see the recent files sorted by the application you used them in, or by file type.

Notice that other than in W***** you will be able to access all "recent files" list of your applications from Workbench. W***** usually stores a small number of entries from any application in its start menu. So if you haven't started a program in awhile, or made heavy use of other programs, that program's recent files are rather sooner than later kicked out of that list and you have to start the application again to get faster access to them.

recent.library is Freeware. That means, feel free to copy it for anybody, but please leave the archive intact and always distribute the guide with the library. If you are a developer and want to use it in your own program, special rules apply. See the Distribution section in the Developer's Guide for more info.

To Index

1.3 Installation

Installing recent.library

To be able to use recent.library, you need at least AmigaOS 2.0.

To install, run the provided script, or to install it manually, enter a shell and type in the following:

```
CD <PATH>                [where <PATH> is the path you extracted the
                           recent.lha archive to]
COPY recent.library LIBS:
MAKEDIR ENVARC:Recent
```

The last step is very important, otherwise no recent lists will be saved.

To Index

1.4 Troubleshooting

Troubleshooting & Bugs

I have tested recent.library with a test program that did almost anything to it that I thought could ever occur, but you never know if there really isn't a bug or two left (after all, it's 100% assembler, so...). Well, there's certainly as the programmer himself is usually the worst tester, and I have no contact to other Amiga fans (heeello... still anybody out there?!?), so I didn't have any testers. However, I think I tested it thoroughly enough to ensure that it will perform fine in everyday use and not cause some serious errors by the dozens. In my test programs, it worked stable

and without errors.

If you discover a bug in recent.library please email or write me. I will remove it as fast as I can. See Contact for how to contact me.

If an application using recent.library should suddenly go crazy and display wrong entries in the recent file menu, or simply crash, try deleting the file in the ENVARC:Recent/ drawer that matches the application's name. Notice that the chance that this might happen is very very slim as recent.library does more checks than some other libraries. It does even check memory pointers before freeing if the memory was really allocated, although there is no way to get to that point without the memory being allocated, etc. NULL-pointers passed by careless application programmers can't irritate it, too. However, I cannot catch EVERYTHING stupid a programmer might throw at it.

To Index

1.5 Future

The Future of recent.library

Yes, it has some, really ;-)

We're living in a time where more and more people jump ship and leave the Amiga. Granted, with Gateway and the new AI president, it doesn't look like a glorious future. I once said, maybe if the Amiga would be finally dead I would go for a Mac, but now that it looks like all hope is lost, I can't stand the thought of leaving "my" Amiga.

In short, you can be sure that I'll be further developing for the Amiga, and that the support, development and bug fixes for recent.library will continue.

However, recent.library seems pretty ok as is; if you are using it, whether as a developer or user, and think that you need additional functionality, please write me. You'll find my address under Contact.

What is currently planned is a tool for Workbech/DOPus that will allow you to access all recent files from all your applications that support recent.library sorted by application or by filetype.

To Index

1.6 Contact

Contact Information

If you need to contact me, be it praise for the library (very welcome) or bug reports (not so welcome but hey, yes, I'll fix it!), write to:

Rüdiger Hanke
Goerdelerstr. 40
48151 Münster
Germany

Email: tomjoad@muenster.de

[To Index](#)

1.7 Using recent.library in your program

Using recent.library in your program

recent.library makes most sense if a broad number of programs are using it. I hope it will make life for the already hard-hitted Amiga users a bit easier. Thus I've decided to make as little restrictions about using the library in your own program as possible.

As you may have read in the User Guide, recent.library is Freeware. I give all programmers of Amiga software, no matter whether Public Domain, Shareware or commercial, the permission to use recent.library in their program and distribute the recent.library binary on the media their software comes on, provided:

- you point out at some point in the documentation or during the installation that the program is using recent.library, and its copyright
- the binary is left unchanged
- you send me a postcard or email that your program is using recent.library. When my webpage is ready, I will list all programs using the library there.

In this case, you can include the binary in the regular install process (you don't need to point out to the user he must install recent.library if it's not on his computer but can simply copy the library into LIBS:) and keep it stored wherever it best fits. Remember to create a ENVARC:Recent directory if necessary. However, if there is still room on the media your software comes on, I'd be grateful if you'd put the full archive into an "Extras" or so directory.

[To Index](#)

1.8 Library Functions

Library Function overview

```
AllocRecentHandle
FreeRecentHandle
AddRecentFile
AddRecentGroup
GetRecentInfo
SaveRecentList
SetRecentSize
GetRecentMenu
CloseRecentHandle
```

Structures

```
RecentInfo
```

[To Index](#)

1.9 AllocRecentHandle

AllocRecentHandle

```
BOOL AllocRecentHandle( STRPTR appname, APTR *recentHandle );
```

Offset: -30 (recent.library)

Input:

```
appname      (a0): Name of your application. It is copied into an
                internal buffer so it can be a temporary string
recentHandle (a1): A pointer to an APTR that will receive the address
                of the allocated handle
```

Output:

TRUE if the function was successful, FALSE otherwise.

Description:

Allocates a handle required for all other library functions. The function will also test if there's a recent file list for your application stored on disk and load it if it exists.

Appname serves two purposes. First, it is the name under which the recent list will be saved by the library, so the name should uniquely identify your program (not just "Texteditor", "Paint program"). At the same time, all restrictions for AmigaDOS file names apply for the string (no more than 32 chars, no ':', '/' etc.)

Second, appname will be used by the later Workbench tool which allows the user to access all recent files from Workbench as the drawer name for recent lists stored by your program. It should thus be conveniently readable to human beings (not all-uppercase or all-lowercase, usual convention is first letter of a word in the name uppercase, and the remaining letters

lowercase). Ideally, write it as you would write it on the package or in the manual.

Library Functions
To Index

1.10 FreeRecentHandle

FreeRecentHandle

```
void FreeRecentHandle( APTR *recentHandle );
```

Offset: -36 (recent.library)

Input:

recentHandle (a1): A pointer to the handle that was obtained through
AllocRecentHandle

Description:

Frees all memory allocated for the handle, and sets your recent handle to NULL so that you don't have an invalid pointer.

Library Functions
To Index

1.11 CloseRecentHandle

CloseRecentHandle

```
void CloseRecentHandle( APTR *recentHandle );
```

Offset: -78 (recent.library)

Input:

recentHandle (a1): A pointer to the handle that was obtained through
AllocRecentHandle

Description:

Saves the current recent list to disc, frees all memory allocated for the handle, and sets your recent handle to NULL so that you don't have an invalid pointer. It is essentially a shortcut for calling both SaveRecentList and FreeRecentHandle.

Library Functions
To Index

1.12 AddRecentFile

AddRecentFile

```
BOOL AddRecentFile( APTR recentHandle, ULONG groupID, STRPTR filename );
```

Offset: -42 (recent.library)

Input:

recentHandle (a0): The handle that was obtained through AllocRecentHandle
groupID (dl): ID of the file group you want to add the file to
filename (al): Name of the file to add to the list

Output:

TRUE if the function was successful, FALSE otherwise.

Description:

Adds the given file to the group in your recent handle. The file name may be relative, but recent.library will try to set a lock on it, so the file must really exist. You need not worry about this as the function will be able to lock any file you can open in your program (you don't need to give it absolute paths), but it is not possible to enter "fantasy" entries. The file will be added into the recent list as the most recently accessed file. If the file already existed in the list, it receives the latest datestamp, so you should call this function also when the user selects a file from the list of recent files!

[Library Functions](#)

[To Index](#)

1.13 AddRecentGroup

AddRecentGroup

```
BOOL AddRecentGroup( APTR recentHandle, ULONG groupID, UBYTE maxEntries );
```

Offset: -48 (recent.library)

Input:

recentHandle (a0): The handle that was obtained through AllocRecentHandle
groupID (d0): ID of the file group you want to add.
maxEntries (dl): Maximum number of entries this group can hold

Output:

TRUE if the function was successful, FALSE otherwise.

Description:

recent.library allows you to divide your application's recent file lists into groups. You can't add files directly to the handle, but rather into specific groups. A group collects files of a certain file type. It allows you to hold different filetypes for different menus (e.g. images and brushes in a paint program, or text files and macros in an editor) with a single handle.

Notice please that you aren't free to set group IDs as you

like or may not simply set it to 0 if you have only one list to take care of in your program. See the list of group IDs for an overview of currently assigned values. If you can't find a group that matches the filetype of the files you want to enter, contact me and tell me what filetype you want to store in the recent list. I will send you your ID via email asap, unless I'm on vacation on the same day. Do not create own IDs by using a number not in the list of group IDs!

Finally, it should be noted that a handle cannot hold more than one group of the same ID. Yet, if you try to add a group that is already contained in the handle, the function will return TRUE. This is because the groups may have been created by AllocRecentHandle already, and you are unable to tell whether it has already opened an old configuration file and created the groups or whether you will have to add the groups yourself. Notice that the number of entries the group can hold is NOT affected by this function if the group already existed.

Library Functions
To Index

1.14 GetRecentInfo

GetRecentInfo

```
LONG GetRecentInfo( APTR recentHandle, ULONG groupID, struct RecentInfo * ↵  
    recentInfo, UBYTE arraySize );
```

Offset: -54 (recent.library)

Input:

recentHandle	(a0):	The handle that was obtained through AllocRecentHandle
groupID	(d0):	ID of the file group you want info about. For more information, consult AddRecentGroup
recentInfo	(a1):	A pointer to an array of RecentInfo structures which will be filled with information. Can be NULL to query the required array size (see description)
arraySize	(d2):	The size of the array of RecentInfo structures

Output:

The function returns the number of structures written into the array. This do not necessarily have to be the number of allocated recent file slots! Do not use array elements beyond this number as they aren't valid.

If a NULL-pointer is passed for recentInfo, then the size of the array required to hold the entire recent data is returned.

Description:

Allows you to obtain the list of most recently used files. The array is sorted with the first element being the most recently used file. Information obtained includes the file name (for displaying in the menu bar), the full name including path (for

accessing the file), and a DOS datestamp of the last file access. Note that a maximum of arraySize structures are filled. If arraySize is smaller than the number of recent entries in the group, the arraySize most recent ones will be returned. To learn how large your array needs to be to hold the full recent file list, pass a NULL-pointer for recentInfo.

Library Functions
To Index

1.15 SaveRecentList

SaveRecentList

```
void SaveRecentList( APTR recentHandle );
```

Offset: -60 (recent.library)

Input:

recentHandle (a0): The handle that was obtained through AllocRecentHandle

Description:

Saves the data stored by the given handle to disk. A "Recent" directory in ENVARC: must exist or the data will not be saved. You do not have to call this function if you close your handle with CloseRecentHandle instead of FreeRecentHandle.

Library Functions
To Index

1.16 SetRecentSize

SetRecentSize

```
BOOL SetRecentSize( APTR recentHandle, ULONG groupID, UBYTE newSize );
```

Offset: -66 (recent.library)

Input:

recentHandle (a0): The handle that was obtained through AllocRecentHandle
groupID (d0): ID of the filegroup for which to change size
newSize (d1): The new size of the filegroup (i.e. numbers of entries it can hold)

Output:

TRUE if the function was successful, FALSE otherwise.

Description:

Use this function to set the number of files the recent list of a certain group can hold. After calling the function, the MenuItem structure obtained from GetRecentMenu for this group must not necessarily be valid any longer.

[Library Functions](#)
[To Index](#)

1.17 GetRecentMenu

GetRecentMenu

```
struct MenuItem *GetRecentMenu( APTR recentHandle, ULONG groupID );
```

Offset: -72 (recent.library)

Input:

recentHandle (a0): The handle that was obtained through AllocRecentHandle
groupID (d0): ID of the filegroup for which to create a menu

Output:

Pointer to MenuItem structure if the function was successful, NULL otherwise.

Description:

This is a small convenience function that generates you menu entries with the current recent files you can attach to your program's menu. Before use you must call the Gadtools function LayoutMenuItemsA. The item returned by the function is a "dummy" item you should not use. The pointer to the relevant menus is in the SubMenu field. This is necessary for the layout process. For more information, see the small tutorial.

Note that the function is somewhat treacherous when the data in the recent list changes. Before adding files to the list, make sure the user cannot access the menu bar (remove it, etc.). After adding files to a group or resizing it, you must call this function again to get a valid menu. The old menu is outdated in the moment you make changes to the group, and may even no longer exist.

[Library Functions](#)
[To Index](#)

1.18 RecentInfo Structure

RecentInfo

```
struct RecentInfo  
{  
    STRPTR  
        ri_Name,  
        ri_Path;  
    struct DateStamp  
        ri_LastAccess;  
};
```

ri_Name - The name of the file
ri_Path - The name of the file, with full path
ri_LastAccess - Standard DOS datestamp holding the time of the last
 access of this file

Library Functions
To Index

1.19 Style Guide

Style Guides

Here's a few things you should obey when using recent.library in your own programs:

The user should be able to set the size of recent lists
Add options in your preferences menu which allow the user to set the size for every group of recent lists you use individually. Changing the size is no hassle, just call the SetRecentSize function.

You should default the size of the recent list to a reasonable number. Good values should be 8-10 for the main file list, and 3-5 for secondary lists. For example, a paint program could default the image file list to a size of eight and the brush and palette lists to a size of four.

Recent lists should be put into sub-menus
If you attach them to the main "File" or "Project" menu in the Windoze way, not every user will thank you. For example, I have a parttime job in a software company where I'm forced (yes, you read right) to set the number of recent workspace entries to 10 and the number of recent files to 15 in the compiler editor. I think you can guess it doesn't look very pretty when I open the Project menu...

Call AddRecentFile() after a file from the recent list was selected
This will give the file the latest datestamp and move it to the top of the recent list.

Never make up own IDs
I can't mention often enough how important it is that you choose a proper ID from the list of IDs and do not make up or add own ones. If you need an ID for a filetype you cannot find in the list of predefined IDs, email me and I'll give you a number and add it to the list. The correct ID is also important for correct localization later.

Do not use more than one handle for your app
Again, the reason is the future plans of allowing the user to access recent files from Workbench by application. In addition to that, ain't it easier to have just one handle?!?

Check on installation if the directory ENVARC:Recent exists
Create it if necessary. Otherwise recent.library won't save recent

lists to disk.

Uninstall

Don't forget in your uninstall script to delete the program's recent file list from ENVARC:Recent.

[To Index](#)

1.20 Developer Help

Developer Help

The archive contains include files for some Amiga languages in the developer directory. You'll have to copy them manually into the include directory of your compiler/assembler. If you are using a language or compiler for which I haven't included developer information, it should be easy to create correct include files for your compiler/language. If you have done so, please send them to me and I'll include them with the next release.

[To Index](#)

1.21 Tutorial

How to implement recent lists in your program - a short tutorial

For the ones who don't want to bother reading function descriptions, here's a quick way how to add recent menus to your program:

First include the recent.library include files:

```
#include "libraries/recent.h"
#include "pragma/recent_lib.h"
```

On startup, open recent.library:

```
if( !(RecentBase = OpenLibrary( "recent.library", 0 )) )
    cleanUp();
```

You now need to allocate a valid RecentHandle:

```
APTR
    recentHandle;

if( !AllocRecentHandle( "CoolEdit", &recentHandle ) )
    cleanUp();
```

"CoolEdit" here is the name of your program. Several things should be kept in mind for the name. See the description of AllocRecentHandle for more information.

recent.library will check now if there's a recent list saved for your program. If it is, it is automatically loaded.

But in case there was none saved, we still can't do anything with it unless we put some groups in it. Make a list of filetypes you want to keep recent lists of in your program. Check out the list of group IDs and add the appropriate IDs to the handle. CoolEdit obviously is a text editor, so we might have ASCII text and macro files:

```
if( !AddRecentGroup( recentHandle, RECENT_GROUP_ASCII, 10 ) )
    cleanUp(); // error
if( !AddRecentGroup( recentHandle, RECENT_GROUP_MACRO, 5 ) )
    cleanUp(); // error
```

If you can't find the filetype you want to store in a recent list in the ID list, contact me. I will send you the ID number you should use. NEVER make up IDs on your own.

The last parameter is the number of recent files the group can hold. It is only used if the group doesn't exist yet. Notice that when the group already exists (e.g. AllocRecentHandle finds a saved list), the function does nothing but nevertheless returns success. Make the values you give AddRecentGroup the default sizes for recent lists. More information about choosing reasonable sizes can be found in the Style Guide.

At this point, we're basically ready to receive files in the list. But first let's see how you put the list into the menu bar (as it might already exist; remember AllocRecentHandle might have loaded one!). So first you create the menu (most conveniently done with CreateMenuA() from gadtools.library). Let's assume it looks like this:

```
struct NewMenu
nm[] = {
    { NM_TITLE, "Project", 0, 0, 0, 0 },
    { NM_ITEM, "About...", 0, 0, 0, 0 },
    { NM_ITEM, NM_BARLABEL, 0, 0, 0, 0 },
    { NM_ITEM, "Open...", 0, 0, 0, 0 },
    { NM_ITEM, "Open recent", 0, 0, 0, 0 },
    { NM_SUB, "Empty", 0, 0, 0, 0 },
    { NM_ITEM, NM_BARLABEL, 0, 0, 0, 0 },
    { NM_ITEM, "Quit", 0, 0, 0, 0 },
    { NM_END, 0, 0, 0, 0 } };
```

```
APTR
visualInfo = GetVisualInfoA( myScreen, NULL );
```

```
menu = CreateMenuA( nm, NULL );
if( !menu )
    cleanUp();
```

```
LayoutMenuA( menu, visualInfo, NULL );
```

Note that you must free the visualinfo when you don't need it anymore. When trying out, I noticed that the function produced

memory leaks under OS3.0 so you should get the VI at the beginning and free it at the end of your program, and not get and free it for every menu layout process.

Now call `GetRecentMenu` which will return you a pointer to menu data you must first layout. Then you can add it to your menu bar. In the `layoutTags` array, you can provide a `TextAttr` structure to change the font in the recent menu to match the font you are using for the rest of the menu if you are using a different font than the screen font. If you are always using the screen font, you can simply omit this tag:

```
struct MenuItem
    *recentItem,
    *recentMenu = GetRecentMenu( recentHandle, RECENT_GROUP_ASCII );
struct TagItem
    layoutTags[] = { { GTMN_Menu, 0 },
                     { GTMN_TextAttr, (ULONG)&myMenuTextAttr },
                     { TAG_END, 0 } };

if( !recentMenu )
    cleanUp();

layoutTags[ 0 ].ti_Data = (ULONG)menu;
LayoutMenuItemsA( recentMenu, visualInfo, layoutTags );
```

To connect the item with the rest of the menu, get the address of the menu item which should hold the submenu. Important: attach the SUBMENU of the recent menu item that was returned by `GetRecentMenu()`. The item itself is only a dummy to help the layout process:

```
recentItem = ItemAddress( menu, FULLMENUNUM( 0, 3, NOSUB ) );
if( recentItem )
{
    if( recentMenu->SubItem )
        recentItem->SubItem = recentMenu->SubItem;
    else
        // Recent list is empty - Disable item
        recentItem->Flags &= ~0-ITEMENABLED;
}

SetMenuStrip( myWindow, menu );
```

Whenever your program opens a file that should appear in a recent list, call after successful processing (you should not add files the user might have selected but that your program cannot read because they're damaged or of the wrong type):

```
ClearMenuStrip( myWindow );
AddRecentFile( recentHandle, RECENT_GROUP_xxx, filename );
```

Where `xxx` stands for the correct group the file belongs to. We clear the menu strip before adding the file because after adding a file to the group or changing the number of entries

a group can hold, the menu returned by `GetRecentMenu()` is not necessarily valid any longer. So after adding the file, call `GetRecentMenu()` again and attach the resulting menu to your menu as shown above.

Sometimes, though, you might not want to use `GetRecentMenu()` for some reason, or the user has selected a file from the recent list and you need the path. In this case, call `GetRecentInfo()`. It fills you an array of `RecentInfo` structures with everything you need to know. The data in the array is sorted in the same order as it is in the menu bar, so if you need the file path to a menu item the user selected, it has the same index in the array as it has in the recent list submenu. As good programs should allow the user to set the number of entries for the recent list, you should not assume a constant value for the array size. You don't have to hold the maximum number of entries in a variable as you can get it by giving the `GetRecentInfo()` function a NULL pointer:

```
int
    maxEntries;
struct RecentInfo
    *recentInfo;

maxEntries = GetRecentInfo( recentHandle, RECENT_GROUP_ASCII, NULL, 0 );
if( maxEntries == -1 )
    puts( "Error!" );
if( maxEntries > 0 )
{
    // List is not empty
    recentInfo = new struct RecentInfo[ maxEntries ];
    GetRecentInfo( recentHandle, RECENT_GROUP_ASCII, recentInfo, maxEntries );

    printf( "Most recently opened file: %s", recentInfo[ 0 ]->ri_Path );
    printf( "Least recently opened file: %s", recentInfo[ maxEntries-1 ]->ri_Path ↵
    );
    delete recentInfo;
}
```

The information returned by this function can also be helpful to build your own menu strips if you don't want to use `GetRecentMenu()`.

Being the good programmer that you are, you will of course include a preferences option to let the user change the number of entries a recent list of a certain filetype may hold. When the user closes the configuration dialog with "OK", call:

```
SetRecentSize( recentHandle, RECENT_GROUP_ASCII, newGroupSize );
SetRecentSize( recentHandle, RECENT_GROUP_MACRO, newMacroSize );
```

Important: The same applies for `SetRecentSize` as earlier for `AddRecentFile`. You must disable the menu bar before calling the function, and rebuild the menu again thereafter with a fresh `GetRecentMenu()` call.

You don't have to save the number of entries somewhere as you can obtain them through `GetRecentInfo()` at any point as shown above. So there's really no reason for not letting the user set the size of his recent list.

Of course, after everything is done, you need to clean up everything. The standard cleanup code for `recent.library` would look like this:

```
CloseRecentHandle( &recentHandle );
CloseLibrary( RecentBase );
```

The first command saves all recent lists your program uses to disk, frees all memory allocated by the library, and sets `recentHandle` to `NULL` (thus the pointer to the handle and not the handle itself in the function call) so that you don't accidentally work with an invalid pointer.

Enjoy using `recent.library`!

To Index

1.22 Group ID List

Group IDs currently defined for `recent.library`

Text-related groups

<code>RECENT_GROUP_ASCII</code>	unformatted ASCII text file
<code>RECENT_GROUP_ANSI</code>	text file with ANSI escape codes
<code>RECENT_GROUP_SCRIPT</code>	AmigaDOS script (like startup-sequence)
<code>RECENT_GROUP_AREXX</code>	ARexx script
<code>RECENT_GROUP_GUIDE</code>	AmigaGUIDE file
<code>RECENT_GROUP_CATALOG</code>	Locale catalog
<code>RECENT_GROUP_DOCUMENT</code>	Non-ASCII document (AmigaWriter, PDF,...)
<code>RECENT_GROUP_MACRO</code>	If your program has own (non-ARexx) ones
<code>RECENT_GROUP_HTML</code>	A hypertext page

Sound-related groups

<code>RECENT_GROUP_SAMPLE</code>	Sound sample (e.g. IFF, WAVE,...)
<code>RECENT_GROUP_SCORE</code>	Musical score (such as SMUS or CMUS)
<code>RECENT_GROUP_MOD</code>	Tracker module (such as OctaMED, ...)

Image-related groups

<code>RECENT_GROUP_IMAGE</code>	A bitmap image (e.g. IFF, JPEG, PNG, ...)
<code>RECENT_GROUP_BRUSH</code>	A brush (essentially an image, too)
<code>RECENT_GROUP_VECTORIMAGE</code>	A vector image (e.g. EPS,...)
<code>RECENT_GROUP_PALETTE</code>	File containing color information
<code>RECENT_GROUP_MASK</code>	A mask (usually a one-plane bitmap image)

Programming-related groups

<code>RECENT_GROUP_PROJECT</code>	A project or make file for compilers
<code>RECENT_GROUP_CSOURCE</code>	A C(++) sourcecode
<code>RECENT_GROUP_ASMSOURCE</code>	An assembler source
<code>RECENT_GROUP_ESOURCE</code>	An E sourcecode

RECENT_GROUP_BASICSOURCE	A Basic sourcecode
RECENT_GROUP_JAVESOURCE	A Java sourcecode
RECENT_GROUP_REBOLSCRIPT	A Rebol script
RECENT_GROUP_INCLUDE	Include file
RECENT_GROUP_OBJECTFILE	An object file
RECENT_GROUP_LINKLIB	A link library
Miscellaneous groups	
RECENT_GROUP_EXECUTABLE	All kinds of executable programs
RECENT_GROUP_PREFS	Preferences file
RECENT_GROUP_CONFIG	Program configuration
RECENT_GROUP_FONT	Font (Amiga-Font, TrueType, Postscript...)
