supra

COLLABORATORS

	TITLE : supra	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 19, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

supra

1.1 supra.doc

```
 ~----~Supra~library~v1.1 (11. Apr 95) ----~

  AddToolType()
 ~FCopy()~
 ~FileType()~
 ~FreeNewImg()~
 ~MakeNewImg()~
 ~MakePath()~
 ~ObtPens()~
 ~RecDirFree()~
 ~RecDirInit()~
 ~RecDirNext()~
 ~RecDirTags()~
 ~RelPens()~
```

1.2 AddToolType

```
NAME
    AddToolType -- Adds or changes a new/existing icon's tooltype (V10)
    (icon)

SYNOPSIS
    tool = AddToolType(diskobj, tooltype)

    char * = AddToolType(struct DiskObject *, char *);

FUNCTION
    This function lets you add a new tooltype to a disk object's
    tooltype list, or change already existing one.
    It is a smart routine that makes dealing with tooltypes very
    straightforward.

    The following is an example table about how a tooltype list gets
    changed based on a provided tool:
```

```
 existing tooltype | provided tooltype | result
-----------------------------------------------------------
     NOGUI         |     (NOGUI)       | (NOGUI)
     (NOGUI)       |     NOGUI         | NOGUI
     NOGUI         |     NOGUI         | NOGUI
     SIZE=10       |     SIZE=15       | SIZE=15
     (SIZE=10)     |     SIZE=15       | SIZE=15
     SIZE=10       |     (SIZE=15)     | (SIZE=15)
     [a new one]   |     DONOTWAIT     | DONOTWAIT [added to a list]
```

INPUTS

diskobj - points to an allocated DiskObject structure (usually created by GetDiskObject() function).

tooltype - points to a new tooltype string to be added to a provided tooltype list

RESULT

tool = pointer to a provided tooltype string if succeeds, otherwise NULL.

EXAMPLES

This example opens a ram:test.info icon and asks a user to enter tooltypes to be added (until user enters 'end').

```
#include <libraries/supra.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/icon_protos.h>
#include <stdio.h>
#include <string.h>

#define filename "ram:test"

struct Library *IconBase = NULL;

struct DiskObject *diskobj;

char icon[50];

main()
{
    key = NULL;
        if (IntuitionBase = OpenLibrary("intuition.library",0)) {
            if (IconBase = OpenLibrary("icon.library",0)) {
                if (diskobj = GetDiskObject(filename)) {
                    do {
                        gets(icon);
                        if (strcmp(icon, "end") == 0) break;
                        AddToolType(diskobj, icon);
                    } while (TRUE);

                    PutDiskObject(filename, diskobj);
                    FreeDiskObject(diskobj);
                    FreeRemember(&key, TRUE);
```

```
                }
            }
        }

    if (IconBase) CloseLibrary(IconBase);
    if (IntuitionBase) CloseLibrary(IntuitionBase);
}


NOTES
    All memory allocations used by AddToolType() will be stored in
    FreeList structure that MUST be allocated right after the provided
    DiskObject structure. If you called your DiskObject by GetDiskObject()
    or GetDiskObjectNew() then FreeList is automaticly appended.
    All allocated memory is freed when you call FreeDiskObject().

    This function requires icon.library to be opened.
```

1.3 FCopy

```
NAME
    FCopy -- copies source file to destination file (V10)
    (dos V36)

SYNOPSIS
    error = FCopy(source, dest, buffer)

    UBYTE = FCopy(char *, char *, LONG);

FUNCTION
    This function works very similar to C:Copy program. It copies
    a source file to a destination file.

INPUTS
    source - pointer to a source file name (with a relative or
             absolute path)
    dest - pointer to a destination file name
    buffer - maximum size of a buffer (in bytes) to be
             allocated for copying. If this buffer is 0, FCopy()
             will try to allocate buffer a size of a source file,
             or the largest memory block available. (this is the
             fastest way).

RESULT
    error - zero if no error. Function may return one of the
    following error definitions:

        FC_ERR_EXIST - Source file does not exist
        FC_ERR_EXAM  - Error during examination of a source file
        FC_ERR_MEM   - Not enough memory availabe
        FC_ERR_OPEN  - Source file could not be oppened
        FC_ERR_READ  - Error while reading a source file
        FC_ERR_DIR   - Source file path is a directory
        FC_ERR_DEST  - Destination file could not be created
        FC_ERR_WRITE - Error while writing to a destination file
```

```
   EXAMPLE

       /* This example will copy a file c:dir to ram: with a new name
        * list.
        */

       UBYTE err;

       if ((err = FCopy("C:Dir", "ram:list", 0)) == 0) {

           no errors...

       } else {
           printf("Error: %d\n", err); /* Error occured during FCopy()
*/

       }

   NOTES
       If an error occurs then a destination file will not be deleted
       if it has already been partly copied.
```

1.4 FileType

```
   NAME
       FileType -- Examines if a file is a directory or a file (V10)

   SYNOPSIS
       type = FileType(filename)

       LONG = FileType(char *);

   FUNCTION
       Will use dos.library's Examine() function to determine
       whether a specified file(path) exists, and if it is a file
       or a directory.

   INPUTS
       filename - pointer to a filename string

   RESULT
       returns 0 if specified file/path does not exist. If < 0, then
       it is a plain file. If > 0 a directory.
       This function actually returns fib_DirEntryType (from
       struct FileInfoBlock).

   EXAMPLE

       type = FileType("SYS:System");

       type will be > 0, which means that "SYS:System" is a dir.
```

1.5 FreeNewImg

```
NAME
    FreeNewImg -- frees memory allocated by MakeNewImg() (V10)

SYNOPSIS
    FreeNewImg(newImage)

    void FreeNewImg(struct Image *);

FUNCTION
    You must free a new created image with this function, when
    it is no longer needed.

SEE ALSO
    MakeNewImg()
```

1.6 MakeNewImg

```
NAME
    MakeNewImg -- Remap an image to any new colours (V10)

SYNOPSIS
    newImage = MakeNewImg(oldImage, palette)

    struct Image * = MakeNewImg(struct Image *, ULONG *);

FUNCTION
    This function creates a new clone image of a provided
    image, and it remaps the new image according to a provided
    pen colour list.
    This is very useful when you need your image to use
    specific colours anywhere in the available palette.
    (e.g. you obtained some free pens from a palette, and
    you want your image to be shown with those pens).
    It is possible to modify an image's pens with PlanePick and
    PlaneOnOff fields (see Image structure), but this has a major
    limitation: most colour combinations are not possible to get.

    If your image has four colours (0,1,2,3), and you want to
    remap these to (0,16,4,7), you simply call this function,
    providing it with the image and a new colour map, and a
    new image will be created for you.

INPUTS
    oldImage - pointer to an Image structure to be remapped
    palette  - pointer to a list of new pens

    A pens list should contain the exact number of pens as
    an old image uses. (2 if image's depth is 1, 4 if image's
    depth is 2, etc.). An image's colour 0 will be remapped to the
    first pen on the list, image's colour 1 will be remapped to
    the second pen on the list and so on.
```

```
RESULT
    newImage - pointer to a newly initialized remapped old image's
               clone. If there is not enough memory, newImage will
               be NULL.
    IMPORTANT: If a new image was created you have to call
    FreeNewImg() to free the allocated memory, when you no longer
    need to use it!

EXAMPLE

    We have a depth 2 image (4 colours), and we want to use pens
    0,16,4,7 instead of 0,1,2,3:


    struct Image OldImage = {
        ....    /* This is a data of our original image */

    struct Image *NewImage;
    ULONG pal[] = {0, 16, 4, 7}; /* The new pen list */

    if (NewImage = MakeNewImg(&OldImage, &pal[0])) {
       DrawImage(rp, NewImage);
       FreeNewImg(NewImage);    /* We will no longer use it */
    }

NOTE
    A new image's depth will change to a depth that can hold
    the largest pen number from a pens list. It does not have
    any smart routine to check if the depth can be optimized
    down, by altering PlanePick and PlaneOnOff, yet.
    Bear in mind that if you provide a pen 255, then a new image's
    depth will be at least 8.

    You MUST free a new image with FreeNewImg() when it's no longer
    needed!

    This function can take much time when remapping larger images
    with more depths.

BUGS
    None found.

SEE ALSO
    FreeNewImg()
```

1.7 MakePath

```
NAME
    MakePath -- Creates all new directories in a path (V10)

SYNOPSIS
    suc = MakePath(path)

    BOOL = MakePath(char *);
```

```
   FUNCTION
       This function creates a whole specified path of directories.
       It works similar to CreateDir() except that it can create
       more subdirs at once. User does not have to care if all
       sub dirs in a specified path already exist or not.

   INPUTS
       path - pointer to a path string to create. A path can be
       relative to a current dir or absolute.

   RESULT
       suc - TRUE if succeeds (path was created). FALSE if a path
       could not be created.

   EXAMPLE

       suc = MakePath("RAM:way/to/many/dirs");

       The above function will try to make all non-existing dirs
       in a path RAM:way/to/many/dirs.

   SEE ALSO
       CreateDir()
```

1.8 ObtPens

```
   NAME
       ObtPens -- Obtain best pens from a list of colors (V10)
       (gfx V39)

   SYNOPSIS
       number = ObtPens(cm, PalTable, PensTable, TagItem)

       ULONG  = ObtPens(struct ColorMap *, ULONG *, ULONG *,
                 struct TagItem *);

   FUNCTION
       This function calls ObtainBestPen() on a list of color
       entries, and puts results into a new pens list.

       It will attempt to find colors in your viewport closest to
       the provided colors list (PalTable).
       This is usefull when you want to use an image with more
       specific colors on a public screen with a sharable palette.

   INPUTS
       cm = colormap
       PalTable - list of RGB entries for each color you want to use.
                  The format of this table is the same as for
LoadRGB32():

              1 Word with the number of colors to obtain
              1 Word with the first color to be obtained
              3 longwords representing a left justified 32 bit RGB
triplet
```

```
            The list is terminated by a count value of 0.

            examples:
              ULONG PalTable[]={2l<<16+1,0,0,0, 0xffffffff,0,0, 0};
                  two entries (black, red); obtains only red one

       PensTable - list of pen numbers on your viewport, obtained by
                   this function. First entry in PensTable will
represent
                   the first color in PalTable, and so on.
                   NOTE that entries in PensTable with count number
lower
                   than the first color to be obtained (provided in
                   PalTable) will be unaffected!

       TagItem - this tagitem will be passed to ObtainBestPen()
function,
                 that is called within ObtPens(). Please see
ObtainBestPen()
                 in order to decide what kind of precision for
obtaining
                 colors you will need. If this is NULL, PRECISION_IMAGE
will
                 be used.

   RESULT
       number = number of obtained colours (always the same as the
first
       Word in PalTable), or 0 if failed. If it succeeds you must call
       RelPens() to free obtained colors.


   EXAMPLES

       The following example will obtain red, green and blue colours in
a
       viewport, and will put obtained pens into pens[] table. pens[0]
will
       be untouched (will remain the same colour as viewport's
background).

       ULONG pal[((4<<16)+1, /* 4 entries, starting with the second one
*/
                   0, 0, 0,          /* black - will ignore this one */
                   0xffffffff, 0, 0, /* red */
                   0, 0xffffffff, 0, /* green */
                   0, 0, 0xffffffff, /* blue */
                   0};

       ULONG pens[4];

       ObtPens(cm, pal, pens, NULL);

       SetAPen(rp, pens[1]);   /* Set the primary pen to red */
       Text(rp, "I'm red!", 8);
```

```
   NOTES
       You MUST call RelPens() to free all obtained colors if ObtPens()
       have succeeded, but you must not call it if ObtPens() returns 0.
       You MUST open graphics library (V39 or higher) before calling
this
       function.

   SEE ALSO
       RelPens(), ObtainBestPen(), LoadRGB32()
```

1.9 RecDirFree

```
   NAME
       RecDirFree -- Unlocks all locked paths (V10)
       (dos V36)

   SYNOPSIS
       void RecDirFree(RecDirInfo)

       void RecDirFree(struct RecDirInfo *);

   FUNCTION
       This function is called internally when error occurs in
       RecDirNext(), so you don't have to call it then!
       You can only call it when you no longer want to use
RecDirNext(),
       before it finishes the scanning process.
       You DO NOT have to call RecDirFree() when you get any error,
       even DN_ERR_END (scanning process complete)!

   INPUTS
       RecDirInfo - pointer to struct RecDirInfo which has been
                    called with RecDirInit()

   RESULT
       none

   SEE ALSO
       RecDirInit(), RecDirNext(), RecDirNextTagList()
```

1.10 RecDirInit

```
   NAME
       RecDirInit -- Initializes recursive files scanning process (V10)
       (dos V36)

   SYNOPSIS
       error = RecDirInit(RecDirInfo)

       UBYTE = RecDirInit(struct RecDirInfo *)

   FUNCTION
```

```
       This function is required to start scanning files through entire
       or partial directory tree. It locks a directory path provided in
       RecDirInfo structure, then files can be examined by calling
       RecDirNext() function. Please see RecDirNext() for more
       explanation on how this is useful.
       You should initialize RecDirInfo by yourself, and you MUST set
       rdi_Path, rdi_Num, and rdi_Pattern.

   INPUTS
       RecDirInfo - pointer to RecDirInfo structure, which should be
       allocated and initialized before RecDirInit() is called.
       You must set its rdi_Path field to starting directory path
       you want to scan.

       Set rdi_Num for maximum number of directories you wish to scan
       into. If you set rdi_Num to 1 it will only scan one level (that
       rdi_Path points to). If you set rdi_Num to -1 it will scan
       unlimited number of subdirectories deep.

       If rdi_Pattern field is non-NULL and points to a string then
       calling RecDirNext will only return files that match the
       pattern string. NOTE that rdi_Pattern should point to a string
       which has been parsed with ParsePattern().

   RESULT
       error - 0 if no error, otherwise returns one of the following
       errors (also see libraries/supra.h):
           RDI_ERR_FILE - Path provided in rdi_Path points to a file
                          not directory.
           RDI_ERR_NONEXIST - Path provided in rdi_Path does not exist.
           RDI_ERR_MEM - not enough memory to execute RecDirInit().

   EXAMPLE
       Please see an example in RecDirNext() function.

   NOTES
       IMPORTANT: You MUST open dos.library before calling
RecDirInit()!
       rdi_Path is a path relative to a current path your program uses.
       That means you can set rdi_Path to "" to scan from current
       directory, or "/" to scan parent directory.

   BUGS
       None found yet.

   SEE ALSO
       RecDirFree(), RecDirNext(), libraries/supra.h
```

1.11 RecDirNext

```
   NAME
       RecDirNext -- Gets information about the next file (V10)
       (dos V36)

   SYNOPSIS
```

```
       error = RecDirNext(RecDirInfo, RecDirFIB);

       UBYTE = RecDirNext(struct RecDirInfo *, struct RecDirFIB *);

   FUNCTION
       Retrieves information about the next file in a scanning process.
       Calling this function  will not provide a list of sorted files
       niether by ASCII order nor by directory levels. That means
       it will scan files as they have been stored on a disk drive.

       The main advantage of using this function from using ExNext()
       is that you don't have to program a recursive scanning routine
       by yourself. You need only to provide lowest directory path,
       how deep into subdirectories you want to scan, and which
       information about files you need to be provided with.
       RecDirNext() will only return files but no directories.
       You are also able to select a matching pattern so that only
       files which match it will be returned.

       Please see RecDirInit() for more info.

   INPUTS
       RecDirInfo - pointer to RecDirInfo structure. You MUST call
       RecDirInit(), providing it with this structure, before calling
       any RecDirNext() function.

       RecDirFIB - pointer to RecDirFIB structure which should be
       previousely allocated. You only set those fields in the
       structure that you want to have information about. Any field
       should point to a variable into which information will be
stored.
       Check "struct RecDirFIB" to see what each field mean.
       All field in RecDirFIB structure that are set to NULL will be
       ignored.

   RESULT
       error - zero if no error. Otherwise one of the following:
           DN_ERR_END - scanning is completed. You should not call
                        any RecDirNext() again.
           DN_ERR_EXAMINE - Failure while examining a file.
           DN_ERR_MEM - not enough memory available to complete
                        the operation.
           IF any error will be resulted, RecDirFree will be called
           internally.


   EXAMPLE
       This example will scan through the entire HD0: disk device, and
       will print for each file: its dir path, its name, its size.


   #include <stdio.h>
   #include <stdlib.h>
   #include <clib/exec_protos.h>
   #include <clib/dos_protos.h>
   #include <libraries/supra.h>
```

```
   struct RecDirFIB rdf;
   struct RecDirInfo rdi;
   char name[30];
   char path[100];
   LONG size;" link "TEXT_INCLUDE:exec/types.h/Main" 35} size;
   LONG err;" link "TEXT_INCLUDE:exec/types.h/Main" 35} err;

   struct DosBase *DosBase;

   void main()
   {
        if (DosBase = (struct DosBase *)OpenLibrary("dos.library",0)) {

           rdi.rdi_Path = "RAM:";  /* from path "RAM:" */
           rdi.rdi_Num = -1;       /* Unlimited subdirs deep */
           rdi.rdi_Pattern = NULL; /* Don't match files for pattern */

           if (RecDirInit(&rdi) == 0) {
               rdf.Path = path;  /* We want to get files' path, name and
size
 */
               rdf.Name = name;
               rdf.Size = &size;
               while ((err = RecDirNext(&rdi, &rdf)) == 0) {
                   printf("%s (%s) %ld\n", path, name, size);
               }

               /* Now check if DN_ERR_END or some other unexpected error
*/
               switch (err) {
                   case DN_ERR_END:
                       printf("Scanning completed\n");
                       break;
                   case DN_ERR_EXAMINE:
                       printf("Error: trouble examining a file\n");
                       break;
                   case DN_ERR_MEM:
                       printf("Error: not enough memory\n");
               }
           }

           CloseLibrary((struct Library *)DosBase);
       } else printf("Cannot open dos.library\n");
   }


   NOTES
       If you want to end scanning earlier you have to call
RecDirFree()!

   BUGS
       none found

   SEE ALSO
       RecDirInit(), RecDirTags(), RecDirFree(), libraries/supra.h
```

1.12 RecDirTags

```
   NAME
       RecDirNextTagList -- Gets information about next file (V10)
       (dos V36)

   SYNOPSIS
       error = RecDirNextTagList(RecDirInfo, RecDirFIB, TagItems)

       UBYTE = RecDirNextTagList(struct RecDirInfo *, struct RecDirFIB
*,
                   struct TagItem *);


       error = RecDirNextTags(RecDirInfo, RecDirFIB, tag1, ...)

       UBYTE = RecDirNextTags(struct RecDirInfo *, struct RecDirFIB *,
                   ULONG tag1, ...);*


   FUNCTION
       This function does the same as RecDirNext() but it provides
       a TagList extension. Any additional tags will override initial
       settings in RecDirFIB structure.

   INPUTS
       RecDirInfo - pointer to RecDirInfo structure which has been
                    called with RecDirInit()
       RecDirFIB  - pointer to initialized and set RecDirFIB structure,
                    or NULL.
                    You can get files' information either by setting
                    variables to this structure before calling
                    RecDirNextTagList(), or by providing TagItems you
                    want.
                    NOTE: If you provide any TagItem then RecDirFIB
will
                    be changed (if it's non-NULL)!

       Tags - The following tags are available:

           RD_NAME - File name will be provided. ti_Data should carry
                     a pointer to a string buffer (char *)
           RD_PATH - Directory path where scanned file is.
           RD_FULL - Full directory path + file name
           RD_SIZE - File lenght in bytes. ti_Data must have a pointer
to
                     a LONG number (LONG *).
           RD_FLAGS - File's protection flags. ti_Data must have a
pointer
                     to LONG.
           RD_COMMENT - File's comment note. ti_Data carries a pointer
to
                     a string buffer.
           RD_DATE - File's DateStamp structure. Function will copy the
                     entire DateStamp structure into struct DateStamp
                     provided in ti_Data field that points to it.
```

```
           RD_BLOCKS - File size in blocks. ti_Data should have a
pointer
                       to LONG
           RD_UID - Owner's UID (not supported with all file systems).
                    ti_Data should have a pointer to UWORD variable.
           RD_GID - Owner's GID. ti_Data has a pointer to UWORD
variable.
           RD_FIB - FileInfoBlock. Function will copy examined file's
                    FileInfoBlock to a provided struct FileInfoBlock
                    via ti_Data (ti_Data has a pointer to an allocated
                    FileInfoBlock structure).

   RESULT
       Same as for RecDirNext()

   EXAMPLE

       See an example for RecDirNext(). You can replace its line

       RecDirNext(&rdi, &rdf);
          *with*
       RecDirNextTags(&rdi, NULL, RD_PATH, path,
                                  RD_NAME, name,
                                  RD_SIZE, &size,
                                  TAG_DONE);
   NOTES
       If RecDirFIB is not NULL, and you provide some tags as well then
       RecDirFIB will be changed. This may change in the future so
       that provided RecDirFIB will not change.

   BUGS
       none found

   SEE ALSO
       RecDirNext(), RecDirInit(), libraries/supra.h
```

1.13 RelPens

```
   NAME
       RelPens -- Release a list of pens obtained by ObtPens (V10)
       (gfx V39)
   SYNOPSIS
       RelPens(cm, PalTable, PensTable)

       void (struct ColorMap *, ULONG *, ULONG *);

   FUNCTION
       This function repeats calls to ReleasePen() in order to
       release all pens obtained by ObtPens().

   INPUTS
       cm = colormap
       PalTable - the same PalTable called with ObtPens()
       PensTable - the same PensTable called with ObtPens()
```

```
   NOTES
       Please DO NOT modify PalTable and PensTable between calling
       ObtPens() and RelPens(). This function uses the first long
       word from PalTable (describing number of entries and starting
       position), and all entries from PensTable (except those entries
       that are lower than a starting position).
       You MUST open graphics library (V39 or higher) before calling
       this function!

   SEE ALSO
       ObtPens(), ReleasePen()
```

1.14 Supra Library

```
 Supra Library

version 1.1 11/Apr/1995
```


Please report any comments to:

    jurev@gea.fer.uni-lj.si

    Jure Vrhovnik
    Langusova 13
    Ljubljana, 61000
    Slovenia

```
REMEMBER:

You have to #include<libraries/supra.h> in your source code when using
any of supra library functions. You have to link your object codes with
lib:supra.lib



FEW HINTS:

You will notice that some functions contain two version numbers by their
names. The one on the right is the actual function version. The one
under a name explains what version and what ROM library you have to open
before calling a function. e.g. (gfx V39) tells you that you have to
open graphics.library version 39 or higher before calling that function.
```