icon

COLLABORATORS			
	TITLE : icon		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 18, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

icon

1.1 icon.doc

```
 AddFreeList()           FreeDiskObject()        GetDiskObjectNew()
 BumpRevision()          FreeFreeList()          MatchToolValue()
 DeleteDiskObject()      GetDefDiskObject()      PutDefDiskObject()
 FindToolType()          GetDiskObject()         PutDiskObject()
```

1.2 icon.library/AddFreeList

```
NAME
    AddFreeList - add memory to a free list.

SYNOPSIS
    status = AddFreeList(free, mem, len)
      D0                  A0    A1   A2

    BOOL AddFreeList(struct FreeList *, APTR, ULONG);

FUNCTION
    This routine adds the specified memory to the free list.
    The free list will be extended (if required).  If there
    is not enough memory to complete the call, a null is returned.

    Note that AddFreeList does NOT allocate the requested memory.
    It only records the memory in the free list.

INPUTS
    free -- a pointer to a FreeList structure
    mem -- the base of the memory to be recorded
    len -- the length of the memory to be recorded

RESULTS
    status -- TRUE if the call succeeded else FALSE;

SEE ALSO
    AllocEntry(), FreeEntry(), FreeFreeList()
```

```
BUGS
    None
```

1.3 icon.library/BumpRevision

```
NAME
    BumpRevision - reformat a name for a second copy.

SYNOPSIS
    result = BumpRevision(newbuf, oldname)
      D0                  A0      A1

    char *BumpRevision(char *, char *);

FUNCTION
    BumpRevision takes a name and turns it into a "copy_of_name".
    It knows how to deal with copies of copies.  The routine
    will truncate the new name to the maximum dos name size
    (currently 30 characters).

INPUTS
    newbuf - the new buffer that will receive the name
             (it must be at least 31 characters long).
    oldname - the original name

RESULTS
    result - a pointer to newbuf

EXAMPLE
    oldname                          newbuf
    -------                          ------
    "foo"                            "copy_of_foo"
    "copy_of_foo"                    "copy_2_of_foo"
    "copy_2_of_foo"                  "copy_3_of_foo"
    "copy_199_of_foo"                "copy_200_of_foo"
    "copy foo"                       "copy_of_copy foo"
    "copy_0_of_foo"                  "copy_1_of_foo"
    "01234567890123456789012345678901" "copy_of_0123456789012345678901"

SEE ALSO

BUGS
    None
```

1.4 icon.library/DeleteDiskObject

```
NAME
    DeleteDiskObject - Delete a Workbench disk object from disk.     (V37)

SYNOPSIS
    result = DeleteDiskObject(name)
      D0                      A0
```

```
    BOOL DeleteDiskObject(char *);

FUNCTION
    This routine will try to delete a Workbench disk object from disk.
    The name parameter will have a ".info" postpended to it, and the
    info file of that name will be deleted.  If the call fails,
    it will return zero.  The reason for the failure may be obtained
    via IoErr().

    This call also updates the Workbench screen if needed.

    Using this routine protects you from any future changes to
    the way icons are stored within the system.

INPUTS
    name -- name of the object (char *)

RESULTS
    result -- TRUE if it worked, false if not.

EXAMPLE
    error=NULL;

    *Check if you have the right library version*

    if (((struct Library *)IconBase)->lib_Version > 36)
    {
            if (!DeleteDiskObject(name)) error=IoErr();
    }
    else
    {
            * Delete name plus ".info" *
    }

    if (error)
    {
            * Do error routine...*
    }

SEE ALSO
    PutDiskObject(), GetDiskObject(), FreeDiskObject()

BUGS
    None
```

1.5 icon.library/FindToolType

```
NAME
    FindToolType - find the value of a ToolType variable.

SYNOPSIS
    value = FindToolType(toolTypeArray, typeName)
      D0                 A0           A1
```

```
    char *FindToolType(char **, char *);

FUNCTION
    This function searches a tool type array for a given entry,
    and returns a pointer to that entry.  This is useful for
    finding standard tool type variables.  The returned
    value is not a new copy of the string but is only
    a pointer to the part of the string after typeName.

INPUTS
    toolTypeArray - an array of strings (char **).
    typeName - the name of the tooltype entry (char *).

RESULTS
    value - a pointer to a string that is the value bound to typeName,
            or NULL if typeName is not in the toolTypeArray.

EXAMPLE
    Assume the tool type array has two strings in it:
        "FILETYPE=text"
        "TEMPDIR=:t"

    FindToolType( toolTypeArray, "FILETYPE" ) returns "text"
    FindToolType( toolTypeArray, "filetype" ) returns "text"
    FindToolType( toolTypeArray, "TEMPDIR" )  returns ":t"
    FindToolType( toolTypeArray, "MAXSIZE" )  returns NULL

SEE ALSO
    MatchToolValue()

BUGS
    None
```

1.6 icon.library/FreeDiskObject

```
NAME
    FreeDiskObject - free all memory in a Workbench disk object.

SYNOPSIS
    FreeDiskObject(diskobj)
                   A0

    void FreeDiskObject(struct DiskObject *);

FUNCTION
    This routine frees all memory in a Workbench disk object, and the
    object itself.  It is implemented via FreeFreeList().

    GetDiskObject() takes care of all the initialization required
    to set up the object's free list.  This procedure may ONLY
    be called on a DiskObject allocated via GetDiskObject().

INPUTS
    diskobj -- a pointer to a DiskObject structure
```

```
RESULTS
    None

SEE ALSO
    GetDiskObject(), PutDiskObject(), DeleteDiskObject(), FreeFreeList()

BUGS
    None
```

1.7 icon.library/FreeFreeList

```
NAME
    FreeFreeList - free all memory in a free list.

SYNOPSIS
    FreeFreeList(free)
                 A0

    void FreeFreeList(struct FreeList *);

FUNCTION
    This routine frees all memory in a free list, and the
    free list itself.  It is useful for easily getting
    rid of all memory in a series of structures.  There is
    a free list in a Workbench object, and this contains
    all the memory associated with that object.

    A FreeList is a list of MemList structures.  See the
    MemList and MemEntry documentation for more information.

    If the FreeList itself is in the free list, it must be
    in the first MemList in the FreeList.

INPUTS
    free -- a pointer to a FreeList structure

RESULTS
    None

SEE ALSO
    AllocEntry(), FreeEntry(), AddFreeList()

BUGS
    None
```

1.8 icon.library/GetDefDiskObject

```
NAME
    GetDefDiskObject - read default wb disk object from disk.       (V36)

SYNOPSIS
    diskobj = GetDefDiskObject(def_type)
```

```
     D0                          D0

    struct DiskObject *GetDefDiskObject(LONG);

FUNCTION
    This routine reads in a default Workbench disk object from disk.
    The valid def_types can be found in workbench/workbench.h and
    currently include WBDISK thru WBGARBAGE.  If the call fails,
    it will return zero.  The reason for the failure may be obtained
    via IoErr().

    Using this routine protects you from any future changes to
    the way default icons are stored within the system.

INPUTS
    def_type - default icon type (WBDISK thru WBKICK).  Note that the
               define 'WBDEVICE' is not currently supported.

RESULTS
    diskobj -- the default Workbench disk object in question

SEE ALSO
    PutDefDiskObject

BUGS
    None
```

1.9 icon.library/GetDiskObject

```
NAME
    GetDiskObject - read in a Workbench disk object from disk.

SYNOPSIS
    diskobj = GetDiskObject(name)
      D0                    A0

    struct DiskObject *GetDiskObject(char *);

FUNCTION
    This routine reads in a Workbench disk object in from disk.  The
    name parameter will have a ".info" postpended to it, and the
    info file of that name will be read.  If the call fails,
    it will return zero.  The reason for the failure may be obtained
    via IoErr().

    Using this routine protects you from any future changes to
    the way icons are stored within the system.

    A FreeList structure is allocated just after the DiskObject
    structure; FreeDiskObject makes use of this to get rid of the
    memory that was allocated.

INPUTS
    name -- name of the object (char *) or NULL if you just want a
            DiskObject structure allocated for you (useful when
```

```
            calling AddAppIcon in workbench.library).

RESULTS
    diskobj -- the Workbench disk object in question

SEE ALSO
    GetDiskObjectNew(), PutDiskObject(), DeleteDiskObject(),
    FreeDiskObject()

BUGS
    None
```

1.10 icon.library/GetDiskObjectNew

```
NAME
    GetDiskObjectNew - read in a Workbench disk object from disk.    (V36)

SYNOPSIS
    diskobj = GetDiskObjectNew(name)
      D0                       A0

    struct DiskObject *GetDiskObjectNew(char *);

FUNCTION
    This routine reads in a Workbench disk object in from disk.  The
    name parameter will have a ".info" postpended to it, and the
    info file of that name will be read.  If the call fails,
    it will return zero.  The reason for the failure may be obtained
    via IoErr().

    Using this routine protects you from any future changes to
    the way icons are stored within the system.

    A FreeList structure is allocated just after the DiskObject
    structure; FreeDiskObject makes use of this to get rid of the
    memory that was allocated.

    This call is functionally identical to GetDiskObject with one
    exception.  If its call to GetDiskObject fails, this function calls
    GetDefDiskObject.  This is useful when there is no .info file for the
    icon you are trying to get a disk object for.  Applications that use
    workbench application windows MUST use this call if they want to
    handle the user dropping an icon (that doesn't have a .info file) on
    their window.  The V2.0 icon editor program is an example of a
    workbench application window that uses this call.

INPUTS
    name -- name of the object (char *) or NULL if you just want a
            DiskObject structure allocated for you (useful when
            calling AddAppIcon in workbench.library).

RESULTS
    diskobj -- the Workbench disk object in question

SEE ALSO
```

```
    FreeDiskObject(), GetDiskObject(), PutDiskObject(), DeleteDiskObject()

BUGS
    None
```

1.11 icon.library/MatchToolValue

```
NAME
    MatchToolValue - check a tool type variable for a particular value.

SYNOPSIS
    result = MatchToolValue(typeString, value)
      D0                        A0        A1

    BOOL MatchToolValue(char *, char *);

FUNCTION
    MatchToolValue is useful for parsing a tool type value for
    a known value.  It knows how to parse the syntax for a tool
    type value (in particular, it knows that '|' separates
    alternate values).  Note that the parsing is case insensitve.

INPUTS
    typeString - a ToolType value (as returned by FindToolType)
    value - you are interested if value appears in typeString

RESULTS
    result - TRUE if the value was in typeString else FALSE.

EXAMPLE
    Assume there are two type strings:
        type1 = "text"
        type2 = "a|b|c"

    MatchToolValue( type1, "text" ) returns TRUE
    MatchToolValue( type1, "TEXT" ) returns TRUE
    MatchToolValue( type1, "data" ) returns FALSE
    MatchToolValue( type2, "a" ) returns TRUE
    MatchToolValue( type2, "b" ) returns TRUE
    MatchToolValue( type2, "d" ) returns FALSE
    MatchToolValue( type2, "a|b" ) returns FALSE

SEE ALSO
    FindToolType()

BUGS
    None
```

1.12 icon.library/PutDefDiskObject

```
NAME
    PutDefDiskObject - write disk object as the default for its type.
```

```
                        (V36)

SYNOPSIS
    status = PutDefDiskObject(diskobj)
      D0                      A0

    BOOL PutDefDiskObject(struct DiskObject *);

FUNCTION
    This routine writes out a DiskObject structure, and its
    associated information.  If the call fails, a zero will
    be returned.  The reason for the failure may be obtained
    via IoErr().

    Note that this function calls PutDiskObject internally which means
    that this call (if sucessful) notifies workbench than an icon has
    been created/modified.

    Using this routine protects you from any future changes to
    the way default icons are stored within the system.

INPUTS
    diskobj -- a pointer to a DiskObject

RESULTS
    status -- TRUE if the call succeeded else FALSE

SEE ALSO
    GetDefDiskObject

BUGS
    None
```

1.13 icon.library/PutDiskObject

```
NAME
    PutDiskObject - write out a DiskObject to disk.

SYNOPSIS
    status = PutDiskObject(name, diskobj)
      D0                   A0      A1

    BOOL PutDiskObject(char *, struct DiskObject *);

FUNCTION
    This routine writes out a DiskObject structure, and its
    associated information.  The file name of the info
    file will be the name parameter with a
    ".info" postpended to it.  If the call fails, a zero will
    be returned.  The reason for the failure may be obtained
    via IoErr().

    As of release V2.0, PutDiskObject (if sucessful) notifies workbench
    han an icon has been created/modified.
```

```
    Using this routine protects you from any future changes to
    the way icons are stored within the system.

INPUTS
    name -- name of the object (pointer to a character string)
    diskobj -- a pointer to a DiskObject

RESULTS
    status -- TRUE if the call succeeded else FALSE

NOTES
    It is recommended that if you wish to copy an icon from one place
    to another than you use GetDiskObject() and PutDiskObject()
    and do not copy them directly.

SEE ALSO
    GetDiskObject(), FreeDiskObject(), DeleteDiskObject()

BUGS
    None
```