

**utility**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> utility		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 19, 2024	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>utility</b>	<b>1</b>
1.1	utility.doc	1
1.2	utility.library/AddNamedObject	2
1.3	utility.library/AllocateTagItems	2
1.4	utility.library/AllocNamedObjectA	3
1.5	utility.library/Amiga2Date	4
1.6	utility.library/ApplyTagChanges	4
1.7	utility.library/AttemptRemNamedObject	5
1.8	utility.library/CallHookPkt	5
1.9	utility.library/CheckDate	6
1.10	utility.library/CloneTagItems	7
1.11	utility.library/Date2Amiga	7
1.12	utility.library/FilterTagChanges	8
1.13	utility.library/FilterTagItems	9
1.14	utility.library/FindNamedObject	10
1.15	utility.library/FindTagItem	10
1.16	utility.library/FreeNamedObject	11
1.17	utility.library/FreeTagItems	11
1.18	utility.library/GetTagData	12
1.19	utility.library/GetUniqueID	12
1.20	utility.library/MapTags	13
1.21	utility.library/NamedObjectName	14
1.22	utility.library/NextTagItem	15
1.23	utility.library/PackBoolTags	16
1.24	utility.library/PackStructureTags	17
1.25	utility.library/RefreshTagItemClones	18
1.26	utility.library/ReleaseNamedObject	18
1.27	utility.library/RemNamedObject	19
1.28	utility.library/SDivMod32	19
1.29	utility.library/SMult32	20

---

1.30	utility.library/SMult64 . . . . .	21
1.31	utility.library/Stricmp . . . . .	21
1.32	utility.library/Strnicmp . . . . .	22
1.33	utility.library/TagInArray . . . . .	23
1.34	utility.library/ToLower . . . . .	23
1.35	utility.library/ToUpper . . . . .	24
1.36	utility.library/UDivMod32 . . . . .	24
1.37	utility.library/UMult32 . . . . .	25
1.38	utility.library/UMult64 . . . . .	26
1.39	utility.library/UnpackStructureTags . . . . .	26

---

# Chapter 1

## utility

### 1.1 utility.doc

```
AddNamedObject ()
AllocateTagItems ()
AllocNamedObjectA ()
Amiga2Date ()
ApplyTagChanges ()
AttemptRemNamedObject ()
CallHookPkt ()
CheckDate ()
CloneTagItems ()
Date2Amiga ()
FilterTagChanges ()
FilterTagItems ()
FindNamedObject ()
FindTagItem ()
FreeNamedObject ()
FreeTagItems ()
GetTagData ()
GetUniqueID ()
MapTags ()
NamedObjectName ()
NextTagItem ()
PackBoolTags ()
PackStructureTags ()
RefreshTagItemClones ()
ReleaseNamedObject ()
RemNamedObject ()
SDivMod32 ()
SMult32 ()
SMult64 ()
Stricmp ()
Strnicmp ()
TagInArray ()
ToLower ()
ToUpper ()
UDivMod32 ()
UMult32 ()
UMult64 ()
```

UnpackStructureTags()

## 1.2 utility.library/AddNamedObject

NAME

AddNamedObject -- add a named object to the given namespace. (V39)

SYNOPSIS

```
success = AddNamedObject(nameSpace, object);  
D0      A0      A1
```

```
BOOL AddNamedObject(struct NamedObject *, struct NamedObject *);
```

FUNCTION

Adds a new item to a NameSpace. If the NameSpace doesn't support duplicate names, a search for a duplicate will be made, and 0 (failure) will be returned. Otherwise, the entry will be Enqueue()ed to the NameSpace.

INPUTS

nameSpace - the name space to add to (NULL for root namespace)  
object - the object to add (If NULL, will return failure)

RESULT

success - whether the operation succeeded. Check this always!

SEE ALSO

AttemptRemNamedObject(), RemNamedObject()

## 1.3 utility.library/AllocateTagItems

NAME

AllocateTagItems -- allocate a tag list. (V36)

SYNOPSIS

```
tagList = AllocateTagItems(numTags);  
D0      D0
```

```
struct TagItem *AllocateTagItems(ULONG);
```

FUNCTION

Allocates the specified number of usable TagItems slots.

Note that to access the TagItems in 'tagList', you should use the function NextTagItem(). This will insure you respect any chaining (TAG\_MORE) and secret hiding places (TAG\_IGNORE) that this function might generate.

INPUTS

numTags - the number of TagItem slots you want to allocate.

RESULTS

---

tagList - the allocated chain of TagItem structures, or NULL if there was not enough memory. An allocated tag list must eventually be freed using FreeTagItems().

SEE ALSO

<utility/tagitem.h>, FreeTagItems(), CloneTagItems()

## 1.4 utility.library/AllocNamedObjectA

NAME

AllocNamedObjectA -- allocate a named object. (V39)

SYNOPSIS

```
object = AllocNamedObjectA(name, tagList);
D0                                A0      A1
```

```
struct NamedObject *AllocNamedObjectA(STRPTR, struct TagItem *);
```

```
object = AllocNamedObject(name, Tag1, ...);
```

```
struct NamedObject *AllocNamedObject(STRPTR, ULONG, ...);
```

FUNCTION

Allocates a NamedObject and initializes it as needed to the name given. This object can then be used as an object in the namespaces. Tags can be given to make an object contain a namespace such that nested namespaces can be built. When the object is allocated, it automatically has one use. If you later wish to release this object such that others may remove it from the namespace you must do a ReleaseNamedObject().

INPUTS

name - name for the object (must not be NULL)

tagList - tags with additional information for the allocation or NULL

TAGS

ANO\_NameSpace - BOOL tag, default FALSE. If this tag is TRUE, the named object will also have a name space attached to it.

ANO\_UserSpace - ULONG tag, default 0. If this tag is non-NULL it defines the size (in bytes) of the user space to be allocated with the named object and that will be pointed to by the no\_Object pointer. This memory is long-word aligned. If no space is defined, no\_Object will be NULL.

ANO\_Priority - BYTE tag, default 0. This tag lets you pick a priority for the named object for when it is placed into a name space.

ANO\_Flags - ULONG tag, default 0. This tag lets you set the flags of the NameSpace (if you allocated one) There currently are only TWO flags. Do \*NOT\* set \*any\* other bits as they are for future use!!! (You can't read them anyway) The flags are:

NSF\_NODUPS - Name space must be unique.

NSF\_CASE - Name space is case sensitive

#### RESULT

object - the object allocated, or NULL for failure. The object is defined as a pointer to a pointer. You can do what you wish with the pointer. (It may be NULL or contain a pointer to memory that you had asked for in the tags.)

#### SEE ALSO

FreeNamedObject(), <utility/name.h>

## 1.5 utility.library/Amiga2Date

#### NAME

Amiga2Date -- fill in a ClockData structure based on a system time stamp (V36)

#### SYNOPSIS

```
Amiga2Date(seconds,result);
           D0      A0
```

```
VOID Amiga2Date(ULONG,struct ClockData *);
```

#### FUNCTION

Fills in a ClockData structure with the date and time calculated from a ULONG containing the number of seconds from 01-Jan-1978 to the date.

#### INPUTS

seconds - the number of seconds from 01-Jan-1978.  
result - a pointer to a ClockData structure that will be altered by this function

#### SEE ALSO

CheckDate(), Date2Amiga()

## 1.6 utility.library/ApplyTagChanges

#### NAME

ApplyTagChanges -- change a tag list based on a second tag list. (V39)

#### SYNOPSIS

```
ApplyTagChanges(list,changeList);
           A0    A1
```

```
VOID ApplyTagChanges(struct TagItem *, struct TagItem *);
```

#### FUNCTION

For any tag that appears in both 'list' and 'changeList', this function will change the ti\_Data field of the tag in 'list' to match the ti\_Data field of the tag in 'changeList'. In effect, 'changeList' contains a series of new values for tags already in

'list'. Any tag in 'changeList' that is not in 'list' is ignored.

#### INPUTS

list - a list of existing tags (may be NULL)

changeList - a list of tags to modify 'list' with (may be NULL)

#### SEE ALSO

<utility/tagitem.h>, FilterTagChanges()

## 1.7 utility.library/AttemptRemNamedObject

#### NAME

AttemptRemNamedObject -- attempt to remove a named object. (V39)

#### SYNOPSIS

```
result = AttemptRemNamedObject(object);
```

```
D0                                A0
```

```
LONG AttemptRemNamedObject(struct NamedObject *);
```

#### FUNCTION

Attempts to remove an object from whatever NameSpace it's in. You must have found the object first (in order to get a use count) before trying to remove it. If the object is in use or is in the process of being removed, this function will return a failure code. If the object is fully removed, the object will then be available to be FreeNamedObject().

#### INPUTS

object - the object to attempt to remove The object must be valid

#### RESULT

success - FALSE if object is still in use (somewhere)  
TRUE if object was removed

#### SEE ALSO

RemNamedObject(), AddNamedObject(), ReleaseNamedObject()

## 1.8 utility.library/CallHookPkt

#### NAME

CallHookPkt -- invoke a Hook function callback. (V36)

#### SYNOPSIS

```
return = CallHookPkt(hook,object,message);
```

```
D0          A0    A2    A1
```

```
ULONG CallHookPkt(struct Hook *,APTR,APTR);
```

#### FUNCTION

Performs the callback standard defined by a Hook structure. This function is really very simple; it effectively performs

a JMP to Hook->h\_Entry.

It is probably just as well to do this operation in an assembly language function linked in to your program, possibly from a compiler supplied library or a builtin function.

It is anticipated that C programs will often call a 'varargs' variant of this function which will be named CallHook. This function must be provided in a compiler specific library, but an example of use would be:

```
result = CallHook(hook,dataobject,COMMAND_ID,param1,param2);
```

The function CallHook() can be implemented in many C compilers like this:

```
ULONG CallHook(struct Hook *hook, APTR object, ULONG command, ... )
{
    return(CallHookPkt(hook,object,(APTR)&command));
}
```

#### INPUTS

hook - pointer to an initialized Hook structure as defined in <utility/hooks.h>

object - useful data structure in the particular context the hook is being used for.

message - pointer to a message to be passed to the hook. This is not an Exec Message structure, but is a message in the OOP sense.

#### RESULTS

return - the value returned by the hook function.

#### WARNING

The functions called through this function should follow normal register conventions unless EXPLICITLY documented otherwise (and they have a good reason too).

#### SEE ALSO

<utility/hooks.h>

## 1.9 utility.library/CheckDate

#### NAME

CheckDate -- checks a ClockData structure for legal date. (V36)

#### SYNOPSIS

```
seconds = CheckDate(date);
```

```
D0                                A0
```

```
ULONG CheckDate(struct ClockData *);
```

#### FUNCTION

Determines if the ClockData structure contains legal date information and returns the number of seconds from 01-Jan-1978 to that date, or 0 if the ClockData structure contains illegal data.

---

**INPUTS**

date - a filled-in ClockData structure

**RESULTS**

seconds - 0 if date is invalid, otherwise the number of seconds from 01-Jan-1978 to the date

**BUGS**

The wday field of the ClockData structure is not checked.

**SEE ALSO**

Amiga2Date(), Date2Amiga()

## 1.10 utility.library/CloneTagItems

**NAME**

CloneTagItems -- copy a tag list. (V36)

**SYNOPSIS**

```
clone = CloneTagItems(original);  
D0          A0
```

```
struct TagItem *CloneTagItems(struct TagItem *);
```

**FUNCTION**

Copies the essential contents of a tag list into a new tag list.

The cloning is such that calling FindTagItem() with a given tag on the original or cloned tag lists will always return the same tag value. That is, the ordering of the tags is maintained.

**INPUTS**

original - tag list to clone. May be NULL, in which case an empty tag list is returned.

**RESULTS**

clone - copy of the original tag list, or NULL if there was not enough memory. This tag list must eventually be freed by calling FreeTagItems().

**SEE ALSO**

<utility/tagitem.h>, AllocateTagItems(), FreeTagItems(), RefreshTagItemClones()

## 1.11 utility.library/Date2Amiga

**NAME**

Date2Amiga -- calculate seconds from 01-Jan-1978. (V36)

**SYNOPSIS**

```
seconds = Date2Amiga(date);
```

---

D0

A0

```
ULONG Date2Amiga(struct ClockData *);
```

#### FUNCTION

Calculates the number of seconds from 01-Jan-1978 to the date specified in the ClockData structure.

#### INPUTS

date - pointer to a ClockData structure containing the date of interest.

#### RESULTS

seconds - the number of seconds from 01-Jan-1978 to the date specified.

#### WARNING

This function does no sanity checking of the data in the ClockData structure.

#### SEE ALSO

Amiga2Date(), CheckDate()

## 1.12 utility.library/FilterTagChanges

#### NAME

FilterTagChanges -- eliminate tags which specify no change. (V36)

#### SYNOPSIS

```
FilterTagChanges(changeList,originalList,apply);
```

A0            A1            D0

```
VOID FilterTagChanges(struct TagItem *, struct TagItem *, ULONG);
```

#### FUNCTION

This function goes through changeList. For each item found in changeList, if the item is also present in originalList, and their data values are identical, then the tag is removed from changeList. If the two tag's data values are different and the 'apply' value is non-zero, then the tag data in originalList will be updated to match the value from changeList.

#### INPUTS

changeList - list of new tags (may be NULL)  
originalList - a list of existing tags (may be NULL)  
apply - boolean specification as to whether the data values in originalList are to be updated to the data values in changeList.

#### EXAMPLE

Assume you have an attribute list for an object (originalList) which looks like this:

```
{ATTR_Size, "large"},
{ATTR_Color, "orange"},
{ATTR_Shape, "square"}
```

If you receive a new tag list containing some changes (changeList), which looks like this:

```
{ATTR_Size, "large"},
{ATTR_Shape, "triangle"}
```

If you call FilterTagChanges(), changeList will be modified to contain only those attributes which are different from those in originalList. All other items will have their tag values set to TAG\_IGNORE. The resulting changeList will become:

```
{TAG_IGNORE, "large"},
{ATTR_Shape, "triangle"}
```

If 'apply' was set to 0, originalList would be unchanged. If 'apply' was non-zero, originalList would be changed to:

```
{ATTR_Size, "large"},
{ATTR_Color, "orange"},
{ATTR_Shape, "triangle"}
```

SEE ALSO

<utility/tagitem.h>, ApplyTagChanges()

## 1.13 utility.library/FilterTagItems

NAME

FilterTagItems -- remove selected items from a tag list. (V36)

SYNOPSIS

```
numValid = FilterTagItems(tagList,filterArray,logic);
D0                A0    A1        D0
```

```
ULONG FilterTagItems(struct TagItem *,Tag *,ULONG);
```

FUNCTION

Removes tag items from a tag list (by changing ti\_Tag to TAG\_IGNORE) depending on whether its ti\_Tag value is found in an array of tag values.

If the 'logic' parameter is TAGFILTER\_AND, then all items not appearing in 'tagArray' are excluded from 'tagList'.

If 'logic' is TAGFILTER\_NOT, then items not found in 'tagArray' are preserved, and the ones in the array are cast out.

INPUTS

tagList - input list of tag items which is to be filtered by having selected items changed to TAG\_IGNORE.

filterArray - an array of tag values, terminated by TAG\_DONE, as specified in the documentation for TagInArray().

logic - specification whether items in 'tagArray' are to be included or excluded in the filtered result.

RESULTS  
numValid - number of valid items left in resulting filtered list.

SEE ALSO  
<utility/tagitem.h>, TagInArray()

## 1.14 utility.library/FindNamedObject

NAME  
FindNamedObject -- find the next object of a given name. (V39)

SYNOPSIS  
object = FindNamedObject(nameSpace, name, lastObject);  
D0                                   A0                   A1           A2  
  
struct NamedObject \*FindNamedObject(struct NamedObject \*, STRPTR,  
                                  struct NamedObject \*);

FUNCTION  
Finds an object and adds to the open count of the object. The object is guaranteed not to be freed until ReleaseNamedObject() is called. The name comparison is caseless, using the current locale string comparison routines.

If name is NULL, then all objects will be matched.

If lastObject is non-NULL, it must be an object from the same NameSpace found on a previous call to FindNamedObject(). It will not be freed by this call. The search will start at the node after lastobject, if non-NULL.

nameSpace is the name space from the named object given or the root name space if NULL is given.

INPUTS  
nameSpace - the name space to search  
name - the name of the object to search for  
lastObject - the starting point for the search or NULL

RESULT  
object - the first match found, or NULL for no match

SEE ALSO  
ReleaseNamedObject(), <utility/name.h>

## 1.15 utility.library/FindTagItem

NAME  
FindTagItem -- scan a tag list for a specific tag. (V36)

SYNOPSIS  
tag = FindTagItem(tagValue, tagList);

---

D0            D0            A0

```
struct TagItem *FindTagItem(Tag, struct TagItem *);
```

#### FUNCTION

Scans a tag list and returns a pointer to the first item with ti\_Tag matching the 'tagValue' parameter.

#### INPUTS

tagValue - tag value to search for  
tagList - tag item list to search (may be NULL)

#### RESULTS

tag - a pointer to the item with ti\_Tag matching 'tagValue' or NULL if no match was found.

#### SEE ALSO

<utility/tagitem.h>, GetTagData(), PackBoolTags(), NextTagItem()

## 1.16 utility.library/FreeNamedObject

#### NAME

FreeNamedObject -- frees a name object. (V39)

#### SYNOPSIS

```
FreeNamedObject(object);
A0
```

```
VOID FreeNamedObject(struct NamedObject *);
```

#### FUNCTION

Free one of a number of structures used by utility.library. The item must not be a member of any NameSpace, and no one may have it open other than yourself. If the object also contained a NameSpace, that namespace must be empty. Any additional space allocated via the datasize parameter for AllocNamedObject() is also released.

#### INPUTS

object - the object to be freed

#### SEE ALSO

AllocNamedObjectA()

## 1.17 utility.library/FreeTagItems

#### NAME

FreeTagItems -- free an allocated tag list. (V36)

#### SYNOPSIS

```
FreeTagItems(tagList);
A0
```

```
VOID FreeTagItems(struct TagItem *);
```

#### FUNCTION

Frees the memory of a TagItem list allocated either by AllocateTagItems() or CloneTagItems().

#### INPUTS

tagList - list to free, must have been obtained from AllocateTagItems() or CloneTagItems() (may be NULL)

#### SEE ALSO

<utility/tagitem.h>, AllocateTagItems(), CloneTagItems()

## 1.18 utility.library/GetTagData

#### NAME

GetTagData -- obtain the data corresponding to a tag. (V36)

#### SYNOPSIS

```
value = GetTagData(tagValue,defaultVal,tagList);
D0          D0          D1          A0
```

```
ULONG GetTagData(Tag,ULONG,struct TagItem *);
```

#### FUNCTION

Searches a tag list for a matching tag, and returns the corresponding ti\_Data value for the TagItem found. If no match is found, this function returns the value passed in as 'default'.

#### INPUTS

tagValue - tag value to search for.  
defaultVal - value to be returned if tagValue is not found.  
tagList - the tag list to search.

#### RESULTS

value - the ti\_Data value for the first matching TagItem, or 'default' if a ti\_Tag matching 'Tag' is not found.

#### SEE ALSO

<utility/tagitem.h>, FindTagItem(), PackBoolTags(), NextTagItem()

## 1.19 utility.library/GetUniqueID

#### NAME

GetUniqueID -- return a relatively unique number. (V39)

#### SYNOPSIS

```
id = GetUniqueID();
D0
```

```
ULONG GetUniqueID(VOID);
```

#### FUNCTION

Returns a unique value each time it is called. This is useful for things that need unique ID such as the GadgetHelp ID, etc. Note that this is only unique for 4,294,967,295 calls to this function. Under normal use this is not a problem. This function is safe in interrupts.

#### RESULT

id - a 32-bit value that is unique.

## 1.20 utility.library/MapTags

#### NAME

MapTags -- convert ti\_Tag values in a list via map pairing. (V36)

#### SYNOPSIS

```
MapTags(tagList,mapList,mapType);  
    A0  A1      D0
```

```
VOID MapTags(struct TagItem *,struct TagItem *,ULONG);
```

#### FUNCTION

Apply a "mapping list" mapList to tagList.

If the ti\_Tag field of an item in tagList appears as ti\_Tag in some item in mapList, overwrite ti\_Tag with the corresponding ti\_Data from the map list.

The mapType parameter specifies how the mapping operation is to proceed, with the following available types:

#### MAP\_REMOVE\_NOT\_FOUND

If a tag in tagList does not appear in the mapList, remove it from tagList.

#### MAP\_KEEP\_NOT\_FOUND

To have items which do not appear in the mapList survive the mapping process as-is.

MapTags() is central to BOOPSI gadget interconnections where you want to convert the tag values from one space (the sender) to another (the receiver).

The procedure will change the values of the input tag list tagList (but not mapList).

You can "filter" a list by passing MAP\_REMOVE\_NOT\_FOUND as mapType, and having the data items in mapList equal the corresponding tags.

You can perform the inverse filter ("everything but") by passing a mapType of MAP\_KEEP\_NOT\_FOUND, and creating a map item for every tag you want to filter out, pairing it with a mapped data value of TAG\_IGNORE.

---

For safety and "order independence" of tag item arrays, if you attempt to map some tag to the value TAG\_DONE, the value TAG\_IGNORE will be substituted instead.

#### INPUTS

tagList - input list of tag items which is to be mapped to tag values as specified in mapList.  
 mapList - a "mapping list" tag list which pairs tag values expected to appear in tagList with new values to be substituted in the ti\_Tag fields of tagList (may be NULL)  
 mapType - one of the available mapping types as defined in <utility/tagitem.h>

#### EXAMPLE

```
/* Consider this source list: */
struct TagItem list[] =
{
  {MY_SIZE,    71},
  {MY_WEIGHT,  200},
  {TAG_DONE,   }
};

/* And the mapping list: */
struct TagItem map[] =
{
  {MY_SIZE,    HIS_TALL},
  {TAG_DONE,   }
};

/* Then after MapTags(list,map,MAP_REMOVE_NOT_FOUND), 'list' will
   become: */
{HIS_TALL, 71},
{TAG_IGNORE, },
{TAG_DONE, }

/* Or after MapTags(list,map,MAP_KEEP_NOT_FOUND), 'list' will
   become: */
{HIS_TALL, 71},
{MY_WEIGHT, 200},
{TAG_DONE,  }
```

#### BUGS

Prior to V39, the mapType parameter did not work. The function always behaved as if the parameter was set to MAP\_KEEP\_NOT\_FOUND.

#### SEE ALSO

<utility/tagitem.h>, ApplyTagChanges(), FilterTagChanges()

## 1.21 utility.library/NamedObjectName

#### NAME

NamedObjectName -- return the name of the object. (V39)

#### SYNOPSIS

```
name = NamedObjectName(object);
```

D0

A0

```
STRPTR NamedObjectName(struct NamedObject *);
```

## FUNCTION

Returns the name of the object passed in...

Note that the name string is passed back as just a pointer to a read-only name. If the object goes away, so does the name.

## INPUTS

object - the object, may be NULL in which case this function returns NULL.

## RESULT

name - pointer to the name string, or NULL if 'object' is NULL.

## SEE ALSO

FindNamedObject(), RemNamedObject()

## 1.22 utility.library/NextTagItem

## NAME

NextTagItem -- iterate through a tag list. (V36)

## SYNOPSIS

```
tag = NextTagItem(tagItemPtr);
```

D0        A0

```
struct TagItem *NextTagItem(struct TagItem **);
```

## FUNCTION

Iterates through a tag list, skipping and chaining as dictated by system tags. TAG\_SKIP will cause it to skip the entry and a number of following tags as specified in ti\_Data. TAG\_IGNORE ignores that single entry, and TAG\_MORE has a pointer to another array of tags (and terminates the current array!). TAG\_DONE also terminates the current array. Each call returns either the next tagitem you should examine, or NULL when the end of the list has been reached.

## INPUTS

tagItemPtr - doubly-indirect reference to a TagItem structure.  
The pointer will be changed to keep track of the iteration.

## RESULTS

nextTag - each TagItem in the array or chain of arrays that should be processed according to system tag values defined in <utility/tagitem.h> is returned in turn with successive calls.

## EXAMPLE

```
Iterate(struct TagItem *tags);
{
    struct TagItem *tstate;
    struct TagItem *tag;
```

```

    tstate = tags;
    while (tag = NextTagItem(&tstate))
    {
    switch (tag->ti_Tag)
    {
        case TAG1: ...
            break;

        case TAG2: ...
            break;

        ...
    }
    }
}

```

#### WARNING

Do NOT use the value of \*tagItemPtr, but rather use the pointer returned by NextTagItem().

#### SEE ALSO

<utility/tagitem.h>, GetTagData(), PackBoolTags(), FindTagItem()

## 1.23 utility.library/PackBoolTags

#### NAME

PackBoolTags -- builds a "flag" word from a tag list. (V36)

#### SYNOPSIS

```

flags = PackBoolTags(initialFlags,tagList,boolMap);
D0          D0          A0          A1

```

```

ULONG PackBoolTags(ULONG,struct TagItem *,struct TagItem *);

```

#### FUNCTION

Picks out the boolean tag items in a tag list and converts them into bit-flag representations according to a correspondence defined by the tag list 'boolMap'.

A boolean tag item is one where only the logical value of the ti\_Data is relevant. If this field is 0, the value is FALSE, otherwise TRUE.

#### INPUTS

initialFlags - a starting set of bit-flags which will be changed by the processing of TRUE and FALSE boolean tags in tagList.

tagList - a TagItem list which may contain several tag items defined to be boolean by their presence in boolMap. The logical value of ti\_Data determines whether a tag item causes the bit-flag value related by boolMap to be set or cleared in the returned flag longword.

boolMap - a tag list defining the boolean tags to be recognized, and the bit (or bits) in the returned longword that are to be set

or cleared when a boolean Tag is found to be TRUE or FALSE in tagList.

#### RESULTS

flags - the accumulated longword of bit-flags, starting with initialFlags and modified by each boolean tag item encountered.

#### EXAMPLE

```
/* define some nice user tag values ... */
enum mytags { tag1 = TAG_USER+1, tag2, tag3, tag4, tag5 };

/* this TagItem list defines the correspondence between boolean tags
 * and bit-flag values.
 */
struct TagItem boolMap[] =
{
    {tag1,      0x0001},
    {tag2,      0x0002},
    {tag3,      0x0004},
    {tag4,      0x0008},
    {TAG_DONE,  }
};

/* You are probably passed these by some client, and you want
 * to "collapse" the boolean content into a single longword.
 */

struct TagItem boolExample[] =
{
    {tag1,      TRUE},
    {tag2,      FALSE},
    {tag5,      Irrelevant},
    {tag3,      TRUE},
    {TAG_DONE,  }
};

/* Perhaps 'boolFlags' already has a current value of 0x800002. */
boolFlags = PackBoolTags(boolFlags,boolExample,boolMap);

/* The resulting new value of 'boolFlags' will be 0x800005. */
```

#### WARNING

In a case where there is duplication of a tag in tagList, the last of the identical tags will hold sway.

#### SEE ALSO

<utility/tagitem.h>, GetTagData(), FindTagItem(), NextTagItem()

## 1.24 utility.library/PackStructureTags

#### NAME

PackStructureTags -- pack a structure with values from taglist. (V39)

#### SYNOPSIS

```
num = PackStructureTags(pack,packTable,tagList);
D0          A0    A1      A2
```

```
ULONG PackStructureTags(APTR,ULONG *,struct TagItem *);
```

#### FUNCTION

For each table entry, a FindTagItem() will be done and if the matching tag is found in the taglist, the data field will be packed into the given structure based on the packtable definition.

#### INPUTS

pack - a pointer to the data area to fill in.

packTable - a pointer to the packing information table.

See <utility/pack.h> for definition and macros.

tagList - a pointer to the taglist to pack into the structure

#### RESULTS

num - the number of tag items packed

#### SEE ALSO

<utility/pack.h>, FindTagItem(), UnpackStructureTags()

## 1.25 utility.library/RefreshTagItemClones

#### NAME

RefreshTagItemClones -- rejuvenate a clone from the original. (V36)

#### SYNOPSIS

```
RefreshTagItemClones(clone,original)
          A0    A1
```

```
VOID RefreshTagItemClones(struct TagItem *,struct TagItem *);
```

#### FUNCTION

If (and only if) the tag list 'clone' was created from 'original' by CloneTagItems(), and if 'original' has not been changed in any way, you can reset the clone list to its original state by using this function.

#### INPUTS

clone - return value from CloneTagItems(original)

original - a tag list that hasn't changed since CloneTagItems()

#### SEE ALSO

<utility/tagitem.h>, CloneTagItems(), AllocateTagItems(), FreeTagItems(), ApplyTagChanges()

## 1.26 utility.library/ReleaseNamedObject

#### NAME

ReleaseNamedObject -- free a named object. (V39)

## SYNOPSIS

```
ReleaseNamedObject(object);
    A0
```

```
VOID ReleaseNamedObject(struct NamedObject *);
```

## FUNCTION

Decrements the open count of the object. If the object has been removed, and the count goes to 0, the remover will be notified that the object is now free.

## INPUTS

object - the object to release. (No action if NULL)

## SEE ALSO

FindNamedObject(), RemNamedObject()

## 1.27 utility.library/RemNamedObject

## NAME

RemNamedObject -- remove a named object. (V39)

## SYNOPSIS

```
RemNamedObject(object, message);
    A0          A1
```

```
VOID RemNamedObject(struct NamedObject *, struct Message *);
```

## FUNCTION

This function will post a request to release the object from whatever NameSpace it is in. It will reply the message when the object is fully removed. The message.mn\_Node.ln\_Name field will contain the object pointer or NULL if the object was removed by another process.

This function will effectively do a ReleaseNamedObject() thus you must have "found" the object first.

## INPUTS

object - the object to remove: Must be a valid NamedObject.  
message - message to ReplyMsg() (must be supplied)

## RESULT

The message is replied with the ln\_Name field either being the object or NULL. If it contains the object, the object is completely removed.

## SEE ALSO

AttemptRemNamedObject(), AddNamedObject(), ReleaseNamedObject()

## 1.28 utility.library/SDivMod32

## NAME

SDivMod32 -- signed 32 by 32 bit division and modulus. (V36)

## SYNOPSIS

```
quotient:remainder = SDivMod32(dividend,divisor);  
D0                D0      D1
```

```
LONG:LONG SDivMod32(LONG, LONG);
```

## FUNCTION

Divides the signed 32 bit dividend by the signed 32 bit divisor and returns a signed 32 bit quotient and remainder.

## INPUTS

dividend - signed 32 bit dividend.  
divisor - signed 32 bit divisor.

## RESULTS

quotient - signed 32 quotient of the division.  
remainder - signed 32 remainder of the division.

## NOTES

Unlike other Amiga library function calls, the utility.library 32 bit math routines do NOT require A6 to be loaded with a pointer to the library base. A6 can contain anything the application wishes. This is in order to avoid overhead in calling them.

In addition, the utility.library math routines preserve all address registers including A0 and A1

## SEE ALSO

SMult32(), UDivMod32(), UMult32(), SMult64(), UMult64()

## 1.29 utility.library/SMult32

## NAME

SMult32 -- signed 32 by 32 bit multiply with 32 bit result. (V36)

## SYNOPSIS

```
result = SMult32(arg1,arg2);  
D0                D0      D1
```

```
LONG SMult32(LONG, LONG);
```

## FUNCTION

Returns the signed 32 bit result of multiplying arg1 by arg2.

## INPUTS

arg1, arg2 - numbers to multiply

## RESULTS

result - the signed 32 bit result of multiplying arg1 by arg2.

## NOTES

Unlike other Amiga library function calls, the `utility.library` 32 bit math routines do NOT require A6 to be loaded with a pointer to the library base. A6 can contain anything the application wishes. This is in order to avoid overhead in calling them.

In addition, the `utility.library` math routines preserve all address registers including A0 and A1

SEE ALSO

`SDivMod32()`, `UDivMod32()`, `UMult32()`, `SMult64()`, `UMult64()`

## 1.30 `utility.library/SMult64`

NAME

`SMult64` -- signed 32 by 32 bit multiply with 64 bit result. (V39)

SYNOPSIS

```
result = SMult64(arg1,arg2);
```

```
D0:D1          D0    D1
```

```
LONG SMult64(LONG,LONG);
```

FUNCTION

Returns the signed 64 bit result of multiplying arg1 by arg2.

INPUTS

arg1, arg2 - numbers to multiply

RESULTS

result - the signed 64 bit result of multiplying arg1 by arg2.

NOTES

Unlike other Amiga library function calls, the `utility.library` 32 bit math routines do NOT require A6 to be loaded with a pointer to the library base. A6 can contain anything the application wishes. This is in order to avoid overhead in calling them.

In addition, the `utility.library` math routines preserve all address registers including A0 and A1

SEE ALSO

`SDivMod32()`, `UDivMod32()`, `UMult32()`, `UMult64()`

## 1.31 `utility.library/Stricmp`

NAME

`Stricmp` -- case-insensitive string comparison. (V37)

SYNOPSIS

```
result = Stricmp(string1,string2);
```

```
D0      A0      A1
```

```
LONG Stricmp(STRPTR,STRPTR);
```

#### FUNCTION

This function compares two strings, ignoring case using a generic case conversion routine. If the strings have different lengths, the shorter is treated as if it were extended with zeros.

#### INPUTS

string1, string2 - strings to be compared

#### RESULTS

result - relationship between string1 and string2

<0 means string1 < string2

=0 means string1 = string2

>0 means string1 > string2

#### NOTES

Whenever locale.library is installed in a system, this function is replaced by language-specific code. This means that depending on which language the user has currently selected, identical pairs of strings may return different values when passed to this function. This fact must be taken into consideration when using this function.

#### SEE ALSO

Strnicmp(), locale.library/StrnCmp()

## 1.32 utility.library/Strnicmp

#### NAME

Strnicmp -- length-limited case-insensitive string compare. (V37)

#### SYNOPSIS

```
result = Strnicmp(string1,string2,length);
```

```
D0      A0      A1      D0
```

```
LONG Strnicmp(STRPTR,STRPTR,LONG);
```

#### FUNCTION

This function compares two strings, ignoring case using a generic case conversion routine. If the strings have different lengths, the shorter is treated as if it were extended with zeros. This function never compares more than 'length' characters.

#### INPUTS

string1, string2 - strings to be compared

length - maximum number of characters to examine

#### RESULTS

result - relationship between string1 and string2

<0 means string1 < string2

=0 means string1 = string2

>0 means string1 > string2

#### NOTES

Whenever locale.library is installed in a system, this function is

replaced by language-specific code. This means that depending on which language the user has currently selected, identical pairs of strings may return different values when passed to this function. This fact must be taken into consideration when using this function.

SEE ALSO  
`Stricmp()`, `locale.library/StrnCmp()`

### 1.33 utility.library/TagInArray

NAME  
`TagInArray` -- check if a tag value appears in an array of tag values.  
(V36)

SYNOPSIS  
`result = TagInArray(tagValue,tagArray);`  
D0                    D0            A0

`BOOL TagInArray(Tag,Tag *);`

FUNCTION  
Performs a quick scan to see if a tag value appears in an array terminated with `TAG_DONE`. Returns `TRUE` if the value is found.

The 'tagArray' must be terminated by `TAG_DONE`. Note that this is an array of tag values, NOT an array of `TagItems`.

INPUTS  
`tagValue` - tag value to search array for in array.  
`tagArray` - a simple array of tag values terminated by `TAG_DONE`.

RESULTS  
`result` - `TRUE` if `tagValue` was found in `tagArray`.

SEE ALSO  
<utility/tagitem.h>, `FilterTagItems()`

### 1.34 utility.library/ToLower

NAME  
`ToLower` -- convert a character to lower case. (V37)

SYNOPSIS  
`char = ToLower(char);`  
D0                    D0

`UBYTE ToLower(UBYTE);`

FUNCTION  
Converts a character to lower case, handling international character sets.

---

**INPUTS**

char - character to be converted.

**RESULTS**

char - lower case version of the input character.

**NOTES**

Whenever locale.library is installed in a system, this function is replaced by language-specific code. This means that depending on which language the user has currently selected, a given character may return different results when passed to this function. This fact must be taken into consideration when using this function.

**SEE ALSO**

ToUpper(), locale.library/ConvToLower()

## 1.35 utility.library/ToUpper

**NAME**

ToUpper -- convert a character to upper case. (V37)

**SYNOPSIS**

```
char = ToUpper(char);
```

```
D0          D0
```

```
UBYTE ToUpper(UBYTE);
```

**FUNCTION**

Converts a character to upper case, handling international character sets.

**INPUTS**

char - character to be converted.

**RESULTS**

char - upper case version of input character.

**NOTES**

Whenever locale.library is installed in a system, this function is replaced by language-specific code. This means that depending on which language the user has currently selected, a given character may return different results when passed to this function. This fact must be taken into consideration when using this function.

**SEE ALSO**

ToUpper(), locale.library/ConvToLower()

## 1.36 utility.library/UDivMod32

**NAME**

UDivMod32 -- unsigned 32 by 32 bit division and modulus. (V36)

---

## SYNOPSIS

```
quotient:remainder = UDivMod32(dividend,divisor);  
D0                D1                D0                D1
```

```
ULONG:ULONG UDivMod32(ULONG,ULONG);
```

## FUNCTION

Divides the unsigned 32 bit dividend by the unsigned 32 bit divisor and returns an unsigned 32 bit quotient and remainder.

## INPUTS

dividend - unsigned 32 bit dividend.  
divisor - unsigned 32 bit divisor.

## RESULTS

quotient - unsigned 32 quotient of the division.  
remainder - unsigned 32 remainder of the division.

## NOTES

Unlike other Amiga library function calls, the utility.library 32 bit math routines do NOT require A6 to be loaded with a pointer to the library base. A6 can contain anything the application wishes. This is in order to avoid overhead in calling them.

In addition, the utility.library math routines preserve all address registers including A0 and A1

## SEE ALSO

SDivMod32(), SMult32(), UMult32()

## 1.37 utility.library/UMult32

## NAME

UMult32 -- unsigned 32 by 32 bit multiply with 32 bit result. (V36)

## SYNOPSIS

```
result = UMult32(arg1,arg2);  
D0                D0    D1
```

```
ULONG UMult32(ULONG,ULONG);
```

## FUNCTION

Returns the unsigned 32 bit result of multiplying arg1 by arg2.

## INPUTS

arg1, arg2 - numbers to multiply

## RESULTS

result - the unsigned 32 bit result of multiplying arg1 by arg2.

## NOTES

Unlike other Amiga library function calls, the utility.library 32 bit math routines do NOT require A6 to be loaded with a pointer to the library base. A6 can contain anything the application wishes. This is in order to avoid overhead in calling them.

---

In addition, the `utility.library` math routines preserve all address registers including A0 and A1

SEE ALSO  
`SMod32()`, `SMod32()`, `UDivMod32()`, `SMod64()`, `UMod64()`

## 1.38 utility.library/UMult64

NAME  
`UMult64` -- unsigned 32 by 32 bit multiply with 64 bit result. (V39)

SYNOPSIS  

```
result = UMult64(arg1,arg2);
D0:D1          D0    D1
```

`ULONG UMult64(ULONG,ULONG);`

FUNCTION  
Returns the unsigned 64 bit result of multiplying arg1 by arg2.

INPUTS  
arg1, arg2 - numbers to multiply

RESULTS  
result - the unsigned 64 bit result of multiplying arg1 by arg2.

NOTES  
Unlike other Amiga library function calls, the `utility.library` 32 bit math routines do NOT require A6 to be loaded with a pointer to the library base. A6 can contain anything the application wishes. This is in order to avoid overhead in calling them.

In addition, the `utility.library` math routines preserve all address registers including A0 and A1

SEE ALSO  
`SMod32()`, `SMod32()`, `UDivMod32()`, `SMod64()`

## 1.39 utility.library/UnpackStructureTags

NAME  
`UnpackStructureTags` -- unpack a structure to values in taglist. (V39)

SYNOPSIS  

```
num = UnpackStructureTags(pack,packTable,tagList);
D0          A0    A1          A2
```

`ULONG UnpackStructureTags(APTR,ULONG *,struct TagItem *);`

FUNCTION  
For each table entry, a `FindTagItem()` will be done and if the

matching tag is found in the taglist, the data in the structure will be placed into the memory pointed to by the tag's ti\_Data. ti\_Data \*must\* point to a LONGWORD.

#### INPUTS

pack - a pointer to the data area to be unpacked  
packTable - a pointer to the packing information table.  
See <utility/pack.h> for definition and macros  
tagList - a pointer to the taglist to unpack into

#### RESULTS

num - the number of tag items unpacked

#### SEE ALSO

<utility/pack.h>, FindTagItem(), PackStructureTags()