AmigaMail

COLLABORATORS

	TITLE : AmigaMail	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 23, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AmigaMail

1.1 I-1: TagItems and Tag Lists

Staff

The implementation of tags is one of the many new features of release 2.0. Tags made it possible to add new parameters to system functions without interfering with the original parameters. They also make specifying parameter lists much clearer and easier.

A tag is made up of an attribute/value pair as defined below (from utility/tagitem.h):

```
struct TagItem
{
    ULONG  ti_Tag;    /* identifies the type of this item */
    ULONG  ti_Data;   /* type-specific data, can be a pointer */
};
```

The ti_Tag field specifies an attribute to set. The possible values of ti_Tag are implementation specific. System tags are defined in the include files. The value the attribute is set to is specified in ti_Data. An example of the attribute/value pair that will specify a window's name is:

```
ti_Tag  = WA_Title;
ti_Data = "My Window's Name";
```

The ti_Data field often contains 32-bit data as well as pointers.

One way tags are passed to system functions is in the form of tag lists. A tag list is an array or chain of arrays of TagItem structures. Within this array, different data items are identified by the value of ti_Tag. Items specific to a subsystem (Intuition, Graphics,...) have a ti_Tag value which has the TAG_USER bit set. Global system tags have a ti_Tag value with TAG_USER bit clear. The global system tabs include:

```
TAG_IGNORE - A no-op.  The data item is ignored.
TAG_MORE   - The ti_Data points to another tag list, to support
```

```
                chaining of TagItem arrays.
    TAG_DONE    - Terminates the tag item array (or chain).
    TAG_SKIP    - Ignore the current tag item, and skip the next n array
                  elements, where n is kept in ti_Data.
```

Note that user tags need only be unique within the particular context of their use. For example, the attribute tags defined for OpenWindow() have the same numeric value as some tags used by OpenScreen(), but the same numeric value has different meaning in the different contexts.

System functions receive TagItems is several ways. One way is illustrated in the Intuition function OpenWindow(). This function supports an extented NewWindow structure called ExtNewWindow. When the NW_EXTENDED flag is set in the ExtNewWindow.Flags field, OpenWindow() assumes that the struct TagItem *Extension field contains a pointer to a tag list.

Another method of passing a tag list is to directly pass a pointer to a tag list, as OpenWindowTagList() does in the following code fragment.

```
    struct TagItem tagitem[3];
    struct Window *window;

    tagitem[0].ti_Tag = WA_CustomScreen;
    tagitem[0].ti_Data = screen;    /* Open on my own screen */
    tagitem[1].ti_Tag = WA_Title;
    tagitem[1].ti_Data = "AmigaMail Test Window";
    tagitem[2].ti_Tag = TAG_DONE;   /* Marks the end of the tag array. */

    /* Use defaults for everything else. Will open as big as the screen.*/

    if (window = OpenWindowTagList(NULL, /* Because all window parameters
                                          * are specified using tags, we
                                          * don't need a NewScreen
                                          * structure                   */
                                   tagitem))
    {
        /* rest of code */
        CloseWindow(window);
    }
```

Notice that window parameters need not be explicitly specified. Functions that utilize tags should have reasonable defaults to fall back on in case no valid attribute/value pair was supplied for a particular parameter. This fall back capability is a useful feature. An application only has to specify the attributes that differ from the default, rather than unnecessarily listing all the possible attributes.

The amiga.lib support library offers an example of another way to pass TagItems to a function. Rather than passing a tag list, the function OpenScreenTags() receives the attribute/value pairs in the argument list, much like printf() receives its arguments. Any number of attribute/value pairs can be specified. The following code fragment illustrates the usage of OpenScreenTags().

```
struct Screen *screen;

screen = OpenScreenTags(NULL, /* Don't bother with NewScreen structure */
                        SA_Overscan, OSCAN_TEXT,
                        SA_Title, "Amiga Mail Screen",
                        TAG_DONE );
```

Tags are not exclusively for use with the operating system; the programmer can implement them as well. The run-time library utility.library contains several functions to make using tags easier. For more information on utility.library, see the appropriate release 2.0 Autodocs.