The ASL Library

by John Orr

The *asl.library* library provides the release 2.0 Amiga OS with system standard file and font requesters. Any application that needs to use a file or font requester should use *asl.library* requesters because they are easy to implement, they are easy for the user to learn, and because they are the same from one application to the next, they make applications more user friendly.

This article requires a general understanding of several Amiga concepts: TagItems and TagLists (see the Utility library Autodocs), Intuition screens and windows, some of the *graphics.library*'s font structures, and AmigaDOS pattern matching.

Opening a Simple File Requester

Using an ASL requester requires the use of three functions:

```
    myrequester = APTR AllocAslRequest( ULONG type, struct TagItem *tagList );
                  void FreeAslRequest( APTR request );
 made_selection = BOOL AslRequest( APTR request, struct TagItem *tagList );
```

An ASL requester is based on a FileRequester or a FontRequester structure. Programs must allocate this structure with AllocAslRequest() (or AllocFileRequest()—see the *asl.library* Autodoc for more on this function). Currently, the type field (from the above prototype) can be one of two values (as defined in <*libraries/asl.h*>), `ASL_FileRequest`, to ask for a FileRequester structure, or `ASL_FontRequest`, to ask for a FontRequester structure. The values in this structure are for read access only. Any changes to them are carried out through *asl.library* function calls. The FreeAslRequest() function deallocates an ASL requester structure.

```
struct FileRequester    {   /* (from <libraries/asl.h>) */
        APTR    rf_Reserved1;
        BYTE    *rf_File;       /* Filename pointer             */
        BYTE    *rf_Dir;        /* Directory name pointer       */
        CPTR    rf_Reserved2;
        UBYTE   rf_Reserved3;
        UBYTE   rf_Reserved4;
        APTR    rf_Reserved5;
        WORD    rf_LeftEdge,rf_TopEdge; /* Preferred window pos */
        WORD    rf_Width,rf_Height;   /* Preferred window size  */
        WORD    rf_Reserved6;
        LONG    rf_NumArgs;     /* A-la WB Args, for multiselects */
        struct WBArg *rf_ArgList;
        APTR    rf_UserData;    /* Applihandle (you may write!!) */
        APTR    rf_Reserved7;
        APTR    rf_Reserved8;
        BYTE    *rf_Pat;        /* Pattern match pointer */
        };                      /* note - more reserved fields follow */
```

Based on the requester structure passed to it, AslRequest() brings up a font or file requester. When the user makes a selection, AslRequest returns a value. If it returns anything but `FALSE` (0), the user selected a file (or a font). AslRequest() returns `FALSE` if the user cancels the requester or the requester failed for some reason.

Both AslRequest() and AllocAslRequest() accept a tag list. This tag list is used to set the values in the FileRequester (or FontRequester) structure. The C code example *SimpleFR.c* at the end of this article uses the following tags (all the ASL tags are defined in *<libraries/asl.h>*):

```
ASL_Hail
ASL_Width
ASL_Height
ASL_LeftEdge
ASL_TopEdge
ASL_OKText
ASL_CancelText
ASL_File
ASL_Dir
```

`ASL_Hail` supplies a requester with a string to placed in the title bar of the requester window. `ASL_Width`, `_Height`, `_LeftEdge`, and `_TopEdge` describe the initial dimensions and position of the requester window. `ASL_OKText` and `ASL_CancelText` provide alternate strings for the ‘‘OK’’ and ‘‘Cancel’’ gadgets. Currently, the size of these gadgets limits the length of the names to about six characters. The two remaining tags from the above list, `ASL_File` and `ASL_Dir`, tags are specific to file requesters. They supply the initial file and directory names for the file requester.

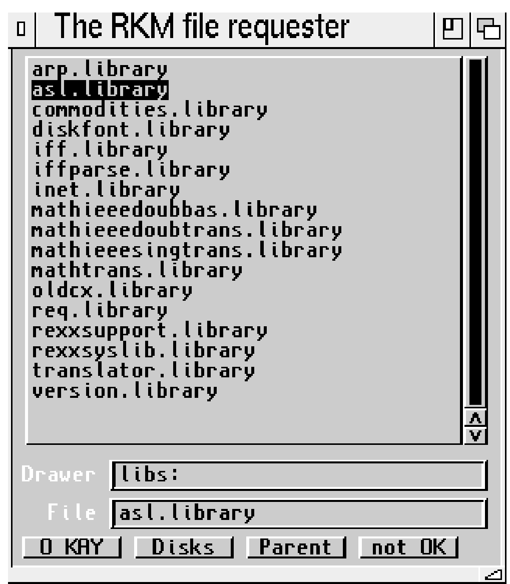

Figure 1 - The SimpleFR.c file requester

For a file requester, if AslRequest() returns anything but `FALSE`, *SimpleFR.c* looks at the FileRequester’s rf_File and rf_Dir fields to get the name and directory of the file the user selected. Note that the requester allows the user to type in a name for the file and directory, which makes it possible for a file requester to return a file and directory that do not (currently) exist. In the case of a save requester (discussed a little later), the requester can create that non-existent directory.

The *simpleFR.c* example (and the rest of the ASL examples for that matter) may be a little misleading because it only uses the requester once. Any application that needs a requester should allocate the requester only once and pop up that requester *every* time the application needs one. If instead, the application allocated, used, and *deallocated* a requester each time it needed one, all of the settings the user left in the requester (like the names the user last typed into the *drawer* or *file* string gadgets) would be reset to their initial values. By reusing a requester, the values stored in it are preserved.

Although retaining previous requester values from one request to another preserves user preferences, it also preserves other values stored in the requester structure. The file and font requester structures contain many private fields that toggle various features of the ASL requesters (those features are set via tags). If a program sets up a file requester with a specific feature, after the requester returns, the file requester structure is still set up with the specific feature. The next time the application wants to use that file requester *without* that specific feature, it must explicitly turn that feature off in the subsequent AslRequest() call.

File Pattern Matching and Multiple Selects

A file requester can filter out certain file and directory entries using *dos.library*'s ‘‘wildcarding’’. Turning on a file requester's ‘‘wildcarding’’ requires using the `ASL_FuncFlags` tag. This tag passes a bit field to a requester to set several options. The options are specific to the type of requester (file or font). The bitmasks are defined in *<libraries/asl.h>*.

The `ASL_FuncFlags`'s `FILF_PATGAD` flag toggles the file requester's ‘‘Pattern’’ gadget. The user can type an AmigaDOS wildcard pattern into this gadget that the requester uses to limit the file names that appear in the requester. An application can supply the ‘‘Pattern’’ gadget with a default pattern matching string using the `ASL_Pattern` tag.

Another feature of the ASL file requester is multiple selection. When an application sets up a file requester, it can set the `FILF_MULTISELECT` flag in the `ASL_FuncFlags` tag that allows the user to select several files at once.

When the user selects several files in the requester, the FileRequester's rf_NumArgs field contains the number of files selected. When the `FILF_MULTISELECT` bit is not set, rf_NumArgs is 0. If rf_NumArgs is not zero, rf_ArgList is a base pointer to an array of WBArg structures (defined in *<workbench/startup.h>*). There is a WBArg structure for each file the user selected.

The example *FilePat.c* illustrates multiple selection.

FilePat.c also introduces a new function and a new tag.

```
APTR AllocAslRequestTags( unsigned long type, Tag Tag1, ... );
BOOL AslRequestTags( APTR request, Tag Tag1, ... );
```

FilePat.c uses AllocAslRequestTags() instead of AllocAslRequest() to allocate and set up the file requester. AllocAslRequestTags() is an *Amiga.lib* function that accepts tag pairs in its parameter list, rather than a pointer to a tag list. AslRequestTags() also accepts tag pairs.

The ASL_Window Tag

An application that uses a custom screen normally wants its requesters to open on its screen. Using the ASL_Window tag, a program can associate a requester with a window so the requester appears on the window's screen. The ASL_Window tag is followed by a pointer to a window structure. ASL_Window works with both file and font requesters.

Normally, a requester associated with a window (using ASL_Window) uses the window's IDCMP port for its communication. An application may not want to share an IDCMP port with the requester. Using the ASL_FuncFlags tag, a program can ask for a requester that creates its own IDCMP port. There are two flags that accomplish this. The first, FILF_NEWIDCMP, is used on file requesters. The other, FONF_NEWIDCMP, is used on font requesters.

The Save Requester

The Save requester is a special type of file requester. The release 2.0 save requester differs from the regular file requester in several ways. The first is its appearance. The color of the text making up the file name list and the color of the backround the text is rendered on are swapped for the save requester. This makes the difference between the save and file requesters visual as well as functional. Another difference, is that a save requester does not allow the user to select by double-clicking on a file name. This prevents the user from accidentally double-clicking, and thus over-writing, the wrong file. The other difference is that if the user types a directory name into the save requester and the directory doesn't exist, the save requester will create that directory (after getting the user's permission via another requester).

The FILF_SAVE flag from the ASL_FuncFlags tag creates a save requester. Note that it does not make sense to have multiple selection in a save requester, so the FILF_SAVE flag overrides the FILF_MULTISELECT flag.

A Directory Requester

A program may not need to process files in a requester, only the directories. There is another variation on *asl.library*'s file requester that allows this. A new tag, ASL_ExtFlags1, toggles this option. ASL_ExtFlags1 passes a bit field like ASL_FuncFlags. One of ASL_ExtFlags1's flags, FIL1F_NOFILES, sets up a requester that has no string gadget for files and displays only directory names in the requester's scrolling list. When AslRequest() (or AslRequestTags()) returns successfully, the FileRequester structure's rf__Dir field contains the name of the directory the user selected.

Currently, there is another flag defined for `ASL_ExtFlags1`, `FIL1F_MATCHDIRS`. If file pattern matching is on (see the `FILF_PATGAD` flag for `ASL_FuncFlags`), setting `FIL1F_MATCHDIRS` tells the file requester to pattern match directory names as well as file names. Of course, if both of `ASL_ExtFlags1`'s flags are set, the requester will only pattern match directory names.

The Font Requester

The ASL library also contains a font requester. The default font requester consists of two scrolling lists, one containing font names, the other containing font sizes. Each scrolling list is accompanied by a string gadget which lets the user type in the name and size of the font they want.

The font size string gadget (actually, an integer gadget) is especially useful because it allows the user to type in a font size that doesn't appear in the font size scrolling list. If the font size the user requests does not already exist in the system, the Amiga can generate the font size the user requests either through bitmap scaling, or, if scalable outline fonts are available, by scaling an outline font.

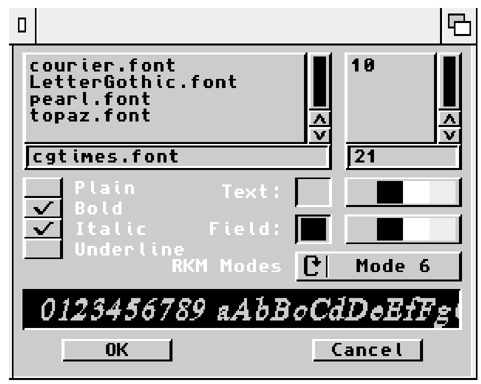

Figure 2 - The FontReq.c font requester

Using the font requester is similar to using the file requester. First, allocate a requester structure. A font requester requires the FontRequester structure:

```
struct FontRequester    {
        APTR    fo_Reserved1[2];
        struct TextAttr fo_Attr;        /* Returned TextAttr            */
        UBYTE   fo_FrontPen;            /* Returned pens, if selected   */
        UBYTE   fo_BackPen;
        UBYTE   fo_DrawMode;
        APTR    fo_UserData;
        };
```

To allocate this structure, use AllocAslRequest() or AllocAslRequestTags() with the first argument set to `ASL_FontRequest`.

There are several tags that are specific to the font requester:

```
ASL_FontName
ASL_FontHeight
ASL_FontStyles
ASL_FontFlags
ASL_FrontPen
ASL_BackPen
ASL_MinHeight
ASL_MaxHeight
ASL_ModeList
```

The first six tags are used to place initial values into the FontRequester structure. They correspond respectively to the FontRequester's fo_Attr.ta_Name, fo_Attr.ta_YSize, fo_Attr.ta_Style, fo_Attr.ta_Flags, fo_FrontPen, and fo_BackPen fields. `ASL_MinHeight` specifies the minimum Y height that the font requester should list. `ASL_MaxHeight` specifies the maximum Y height that the font requester should list. Note that these tags only limit the range of font sizes that the font requester displays, the user is free to type in *any* value. The `ASL_ModeList` tag specifies the strings used for the font requester's drawing mode gadget. This is discussed later in the chapter.

The are several options available to the font requester that are toggled by setting flags with the `ASL_FuncFlags` tag. These flags are:

```
FONF_FRONTCOLOR
FONF_BACKCOLOR
FONF_STYLES
FONF_FIXEDWIDTH
FONF_DRAWMODE
```

A simple font requester (one without any of the above `FONF_` flags set) only lets the user choose a font and a Y size. Setting the flags above adds options to the font requester. `FONF_FRONTCOLOR` and `FONF_BACKCOLOR` add color selection gadgets to the requester, one for choosing a font's foreground color (labeled ‘‘Text’’ on the requester) and the other for choosing the backround color (labeled ‘‘Field’’). The font requester records the user's setting in the FontRequester's fo_FrontPen and fo_BackPen fields. `FONF_STYLES` sets up several gadgets to choose the style of the font (bold, italics, underline). The font requester saves these setting in the fo_Attr.ta_Style bit field according to the style flags defined in *<graphics/text.h>*. `FONF_FIXEDWIDTH` limits the font name display to fixed width (non-proportional) fonts (note that this does not prevent the user from typing in a proportional font name). `FONF_DRAWMODE` adds a cycle gadget to the font requester so the user can choose the draw mode. The draw mode is saved in the requester's fo_DrawMode field. The number stored there corresponds to the draw mode's position in the gadget's cycle.

The draw mode cycle gadget initially is labeled ‘‘Mode’’ and has three elements in its cycle: ‘‘JAM1’’, ‘‘JAM2’’, and ‘‘Complement’’. These yield a result of 0, 1, and 2, respectively. It is possible to change the names and number of draw modes with the `ASL_ModeList` tag. This tag accepts a pointer to an array of strings. The first string replaces ‘‘Mode’’ as the label for the draw mode cycle gadget. The strings that follow replace the elements of the cycle gadget. The last entry in the array has to be `NULL` to tell the requester where the list of entries ends.

The *FontReq.c* example illustrates how to use a font requester.

The `ASL_HookFunc` Tag

The `ASL_HookFunc` tag passes an ASL requester a pointer to a custom function. The requester can use this function for two purposes. The first is to determine if the requester should display a particular file or font name. The other purpose is to process messages that the requester receives at its IDCMP port that are not meant for the requester. Some `ASL_FuncFlags` flags toggle these options:

```
FILF_DOWILDFUNC
FONF_DOWILDFUNC
FILF_DOMSGFUNC
FONF_DOMSGFUNC
```

The `FILF_DOWILDFUNC` and `FONF_DOWILDFUNC` flags cause a requester to call the hook function for every file or font entry. The requester displays the file or font name only if the hook function tells it to. For a file requester, if the hook function returns a *zero*, the file requester will display the file name. For a font requester, if the hook function returns *anything but zero*, the font requester will display the font name and size. Note that if the `DOWILDFUNC` function changes the current directory, it must restort the current directory before it exits. Neglecting to restore the current directory can confuse DOS.

The `FILF_DOMSGFUNC` and `FONF_DOMSGFUNC` flags cause a requester to call the hook function when it receives certain IntuiMessages at the IDCMP port it shares with a window (see the `ASL_Window` tag). The requester passes on IntuiMessages not meant for the requester. The hook function is responsible for returning a pointer to the IntuiMessage. The requester will take care of replying to the message.

A requester passes three parameters to the hook function:

```
ULONG MyHookFunc(ULONG type, CPTR object, CPTR AslRequester)
```

If MyHookFunc() is called from a file requester doing `DOWILDFUNC`:

type = `FILF_DOWILDFUNC`
object is a pointer to an AnchorPath structure (from <*dos/dosasl.h*>)
AslRequester is a pointer to the FileRequester that called the hook function
(Return a zero to display this file)

Note that the AnchorPath structure is a *dos.library* structure used in pattern matching. See *dos.library* documentation for more details.

Referencing the specific file or directory from within the `DOWILDFUNC` hook function is a little tricky. For a file requester, the rf_Dir field is not guaranteed to contain the name of the directory being scanned. The correct way to find the file and directory name is from the AnchorPath structure passed to the `DOWILDFUNC`. The structure contains a FileInfoBlock structure called ap_Info that contains the file name (fib_FileName). The AnchorPath structure's ap_Current field contains a pointer to an AChain structure (from <*dos/dosasl.h*>) which contains a field called an_Lock. This is a lock on the directory being scanned.

If MyHookFunc() is called from a font requester doing `DOWILDFUNC`:

type = `FONF_DOWILDFUNC`
object is a pointer to a TextAttr structure (from <*graphics/text.h*>)
AslRequester is a pointer to the FontRequester that called the hook function
(Return non-zero to display this particular font size)

If MyHookFunc() is called from a *file or font* requester doing `DOMSGFUNC`:

type = `FILF_DOMSGFUNC`
object is a pointer to the IntuiMessage for the function to process
AslRequester is a pointer to the *FileRequester or FontRequester* that called the hook function
(Return the pointer to the IntuiMessage)

Notice that it is possible for a requester to use both `DOWILDFUNC` and `DOMSGFUNC` at the same time. The hook function has to differentiate between the two cases by testing the type passed to it. It is not possible for font and file requester to share a hook function for a `DOWILDFUNC`, because `FILF_DOWILDFUNC` is defined to be the same value as `FONF_DOWILDFUNC`, so the hook function cannot tell if object (from the prototype above) is a pointer to an AnchorPath structure or a pointer to a TextAttr structure. It *is* possible for font and file requesters to share one hook function for `DOMSGFUNC` (even though `FILF_DOMSGFUNC` and `FONF_DOMSGFUNC` are equal) because, in this case, font and file requesters both call the hook function in the same manner.

The example *FileHook.c* illustrates the use of a hook function for both `DOWILDFUNC` and `DOMSGFUNC`.