Menu-Based Browser API to Prospero

Draft of 5 July 1993
Document Revision No. 0.3

Steven Seger Augart B. Clifford Neuman Kwynn Buess Information

Sciences Institute
University of Southern California

This work was supported in part by the National Science Foundation (Grant No. CCR-8619663), the Washington Technology Center, Digital Equipment Corporation, and the Defense Advance Research Projects Agency under NASA Cooperative Agreement NCC-2-539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any of the funding agencies. The authors may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, California 90292-6695, USA. Telephone +1 (310) 822-1511, email `info-prospero@isi.edu`.

Contents

1 INTRODUCTION

The menu-based browser API is a description of interfaces to the PROSPERO library. The functions and variables described here will eventually be prototyped in the standard Prospero include file `p_menu.h`; since the menu browser client and this API are still under development, that include file is currently located in `user/menu` in the Prospero sources, along with the Prospero menu browser client.

2 Data Structures

The data structure *VLINK* is described in the include file `pfs.h`. The user of this API needs to know only the following facts about it:

- A *VLINK* structure has members `next` and `previous`. These members are only meaningful when working with a linked list of *VLINK structures*; they point to the next and previous member of the linked list. The `previous` of the head of the list points to the tail of the list. The `next` of the tail of the list is a null pointer.

- The `APPEND_ITEM(`*VLINK new*`,` *VLINK head*) macro, defined in `list_macros.h`, appends *new* to a doubly-linked list headed by the vlink *head*. *head* is an initialized variable which points to an already existing list of *VLINK*s. It can be initialized to the empty list by setting its value to the null pointer.

- The `EXTRACT_ITEM(`*VLINK item, VLINK head*) macros, also defined in `list_macros.h`, removes the item *item* from the doubly-linked list headed by *head*. If ITEM is not a member of the list, the results are not defined. The extracted item will NOT be freed. `EXTRACT_ITEM` will reset *head* if necessary. This is useful for extracting a single item for future use from a linked list of *VLINK*s and then running `vllfree()` on the list.

The data structure *TOKEN* is also used by this API interface. It obeys the same doubly linked list convention that *VLINK* does.

3 Error Reporting

```
extern char *m_error;
```

All of the functions described in this file set the global variable `m_error` to signal success or failure. Upon failure, they will return with `m_error` set to a pointer to a string which is an error message. Upon success, they will return with `m_error` set to a *NULL* pointer.

4 Library Calls

4.1 *VLINK* `m_top_menu(`*void*)

`m_top_menu()` takes no arguments. It returns a single *VLINK*, which is a link to the first menu to be displayed to the user. You then call m_get_menu() to get the contents of that menu.

4.1.1 Implementation

Return a vlink whose host and hsoname are set from the VSWORK_HOST and VSWORK_FILE environment variables. The NAME of this vlink (the menu title that will be displayed) should for now be the last component of the VSNAME environment variable. Later we will talk about how to start up the browser when somebody is not yet VFSETUP to any virtual system.

4.1.2 Additional ways to get top menu — EXCEPTION TO API

Some menu browsers allow the top menu to be specified on the command line, using either native information or a directory name in the currently active virtual system. At the moment, no API functions are specified to handle this situation, and the PFS library must be called directly. If the top menu is specified as a directory name in the currently active virtual system, just use rd_vlink() for now. If it was specified with native information, vlalloc() a new link, set its HSONAME to the native hsoname, its HOST to the native host, and use rd_vlink(). Eventually, we will design an API function or two to handle getting the top menu by Prospero file name or by host and hsoname.

4.2 *VLINK* `m_get_menu(`*VLINK* `menu)`

`m_get_menu()` is given a *VLINK* to a menu as its argument. It will return an ordered list of *VLINK*s, each corresponding to an item in the menu.

4.2.1 Implementation

This orders the VLINKs according to the COLLATION-ORDER attribute.

When asking for attributes to be returned in the underlying Prospero call, please specify that you want MENU-ITEM-DESCRIPTION+COLLATION-ORDER+ACCESS-METHOD to be returned. (Of course, do this only if you are using a PFS library interface that allows you to specify which attributes you are requested; at the moment, none of the interfaces allow this.) This will help us be compatible with possible future changes to the way the server works.

The 1st two attributes are ones we need to display the menu. Asking for the ACCESS-METHOD right away is also a good idea, since if you don't support an access-method for a file, you can choose show on the menu either that it is unreachable or not display it at all.

4.3 *char ** `m_item_description(`*VLINK vl*)

This returns a string to be displayed as a description for the menu item associated with VLINK. This is a pointer to data that may be overwritten on the next call to `m_item_description()`, but not before.

If *vl* is a link to a sub-menu, then, when that sub-menu is displayed, most clients will also use the string returned by `m_item_description()` as a title for that sub-menu.

4.3.1 Implementation

Look at the MENU-ITEM-DESCRIPTION attribute associated with *vl*. If that fails,look at *vl*'s `name` member.

4.4 *int* `m_class(`*VLINK vl*)

Return a CLASS for the object pointed to by the link. The class says what you can do with an object (view it, read it, run a search through it, open it up as a submenu, use it to connect to another service, etc.) These classes are constants defined in the API.H file. They are: M_CLASS_UNKNOWN (must have a value of 0), M_CLASS_MENU, M_CLASS_DOCUMENT, M_CLASS_SEARCH, M_CLASS_PORTAL, M_CLASS_SOUND, M_CLASS_IMAGE, M_CLASS_DATA, AND M_CLASS_VOID.

4.4.1 Implementation

This is derived from the value of the OBJECT-INTERPRETATION attribute. We are working on a new portable interface to this function which lets the browser specify whether it knows how to display/retrieve/access particular types and particular subtypes. The current implementation of the api returns M_CLASS_DATA for any types that it recognized but cannot perform an appropriate complex operation on (e.g., if the OBJECT-INTERPRETATION is a SOUND but you have no sound player), then return M_CLASS_DATA, and any unknown types are returned as M_CLASS_UNKNOWN.

4.5 *TOKEN* `m_interpretation`(*vLINK vl*)

This will return the sequence that is the value of the OBJECT-INTERPRETATION attribute. You will call m_interpretation() for information that will let you actually display the file. This is used internally by m_class(), too.

4.5.1 Implementation

Warning: The current api implementation does not fully meet this specification for m_interpretation().

If no OBJECT-INTERPRETATION attribute is present, we look at the *vl*'s `target` member and see if it is `DIRECTORY`, `EXTERNAL`, or `FILE`, and we also perform some simple tests on the suffix of a file to check whether it is likely to be a binary or text file. If it ends in `.gif,` it's IMAGE GIF. If it ends in `.ps` or `.PS,` it's DOCUMENT POSTSCRIPT. If it ends in `.Z,` it's EMBEDDED COMPRESS, then strip off the `.Z` and try again for the rest of the object interpretation. If it ends in `.z` or `.gz,` it's EMBEDDED GZIP. Otherwise, assume it's DOCUMENT TEXT ASCII.

The OBJECT-INTERPRETATION attribute is further defined in the `working-notes` subdirectory of the menu sources.

4.6 *FILE ** `m_fopen_file`(*VLINK vl*)

This opens the file referenced by *vl* in read-only mode and returns a standard IO library *FILE* pointer to it, which is then read from and manipulated in accordance with the stdio library routines.

4.6.1 Implementation

This is just an interface to `pfs_fopen()`.

4.7 *int* `m_open_file`(*VLINK vl*)

This opens the file referenced by *vl* in read-only mode and returns a standard UNIX integer file descriptor referring to it. This descriptor can then be manipulated in accordance with the standard conventions.

4.7.1 Implementation

This is just an interface to `pfs_open()`.

4.8 *void* `vlfree`(*vl*), *void* `vllfree`(*vl*), *VLINK* `vlcopy`(*VLINK vl, int r*)

`vlfree()` frees the vlink *vl*. It should be called on the link returned by `m_top_menu()` when the application no longer has a use for it. `vllfree` frees a linked list of *vlink*s headed by *vl*. It should be called on the list returned by `m_get_menu()` when the application no longer has a need for it.

`vlcopy()` returns a copy of *vl*, with the `next` and `previous` members set to the null pointer. *r* should always be zero.

4.8.1 Implementation

These three functions are already in libpfs; you don't need to implement them.