The Prospero Protocol
Version 5

Draft of 12 June 1993

Document Revision No. 0.3

B. Clifford Neuman   Steven Seger Augart

Information Sciences Institute
University of Southern California

Digital copies of the latest revision of this document may be obtained through Prospero as /papers/subjects/operating-systems/prospero/doc/protocol.PS.Z, in the #/INET/EDU/ISI/swa virtual system, or through Anonymous FTP from PROSPERO.ISI.EDU as /pub/prospero/doc/prospero-protocol.PS.Z

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

Chapter 1

Introduction

This document describes version 5 of the Prospero protocol. Communication with directory servers uses a reliable delivery protocol described in Appendix B. Requests and responses are human readable commands and multiple commands may be sent in a single message.

Prospero is implemented on top of a message-based protocol to reduce the overhead that would otherwise be incurred when establishing connections to multiple directory servers. The decision to use a message protocol directly, instead of through a higher level mechanism such as remote procedure call, was made for reasons of portability. We did not want Prospero to depend on another protocol, software package, or other resource, unless that resource was almost universally supported by every computer on the Internet.

The use of humanly readable commands in the protocol has several advantages. It eliminates problems with byte ordering, and it makes future changes or additions to the protocol easier to incorporate while maintaining compatibility across versions; new commands or additional options to existing commands can be easily added.

The ability to request multiple operations in a single message improves performance, and it provides a simple way to request that a collection of operations (on a single server) be performed atomically.

The concept of an "attribute" appears frequently in the following command description. For a discussion of attributes, read appendix A of this document.

1.1 Additional Documents

The Prospero documentation series will include the following:

Prospero Protocol Specification You're reading it. Right now, this is the most current document we have, but it will eventually be re-targeted towards a more specialized audience when the other guides in the series are written and revised.

Prospero System User's Guide This will describe the general features of the Prospero system that will be common to almost all user interfaces, such as ACL types and filters.

Prospero Command-Line Client User's Guide Describes the command-line client package; this is the client that's been shipped with all Prospero releases.

Prospero Gopher Client User's Guide This describes the Gopher menu-based client package, which is in progress.

Prospero Programmer's Manual This describes the `pfs` and `pcompat` libraries, which Prospero applications programmers use to interface with Prospero.

Chapter 2

Command Introduction

2.1 Spaces, quoting, and line-feeds

Prospero messages are divided into lines. Lines are divided into tokens. Commands and lines sent in response are separated by an unquoted ASCII <**LF**> character.

Within a line, tokens are separated by unquoted horizontal whitespace (one or more ASCII spaces and/or tabs). Client and server implementations are not required to accept lines which begin or end with horizontal whitespace, but are encouraged to do so. Similarly, implementations are encouraged to accept <**CR**> or <**CR**><**LF**> as alternative acceptable line terminators, but are not required to do so.

Special characters within a command token, or null tokens, can be quoted using single quotes (`'`). While inside quotes, a quote can be included by doubling it. The only character that cannot be quoted is the ASCII null character (character code zero; `'\0'` for C programmers). This is really a limitation of the server implementation, which passes Prospero packets, commands, and tokens around internally as null-terminated C strings.[1]

Any token may be quoted (although most will not need quoting), except for literal command tokens (`VERSION`, `ID`, etc.). Also, the <**multi-component**> token uses the quoting system in a special way — see the `LIST` command for details.

2.2 Options

Many Prospero Protocol commands take options. If multiple options are to be specified for a single command, the options are separated by a "`+`". If no options are specified, then the null string should be sent to indicate this. The null string will need to be quoted, like so: `''`

2.3 Metasyntax

In this document, we will use some meta-syntactic features to describe commands. Literal tokens (such as `LITERAL-TOKEN`) appear in a typewriter font. Non-terminals look like this: <**non-terminal**>. The line-feed, carriage-return, and space characters are represented as <**LF**>, <**CR**>, and <**SPC**>,

[1] If you wish to send binary data around, we recommend that you encode it into a null-less form using the `pfs` library's `binencode()` and `bindecode()` routines.

respectively. Alternation (choice among two or more possibilities) looks like: (<**option1**> *or* <**option2**>)

We also use the following meta-syntactic constructs:

Metasyntax used to describe commands		
Start	End	Meaning
[	]	One or zero repetitions.
[	]*	Zero or more repetitions.
[	]+	One repetition or more.

2.4 Objects

A Prospero object is anything to which a normal link (<**link-type**> = `L`) can be made, except for `EXTERNAL` objects.

2.4.1 Base Type

Every Prospero object has one or more base types associated with it. All Prospero objects have a base type of OBJECT. This means that they can have attributes attached to them. In addition, objects with a base type of FILE can store data, and objects with a base type of DIRECTORY can store directory information.

The base type of a Prospero object is accessible through the BASE-TYPE intrinsic attribute. The primitive value for BASE-TYPE is `OBJECT`. If an object's BASE-TYPE attribute has a value of `OBJECT+FILE+DIRECTORY`, then the object can have attributes attached to it, can contain data, and can contain links. As a shorthand, since all objects have `OBJECT` as one of their base types, an object with more than one base type can have the `OBJECT` implied, so that the BASE-TYPE value `OBJECT+FILE+DIRECTORY` is equivalent to the value `FILE+DIRECTORY` and the value `FILE` is equivalent to the value `FILE+OBJECT`. The order of values in the BASE-TYPE attribute is unimportant, so `DIRECTORY+FILE` is equivalent to `FILE+DIRECTORY`.

In the rest of this document, when we say "directory,", we mean "an object which has `DIRECTORY` as part of its BASE-TYPE." When we say "file," we mean "an object which has `FILE` as part of its BASE-TYPE."

One may use `EDIT-OBJECT-INFO` to edit the BASE-TYPE attribute to remove the `FILE` type from it only if the file is empty (zero length), and one may remove the `DIRECTORY` type from it only if the directory is empty (contains no links). To add a type to the BASE-TYPE, one must use the `ADD` option to `CREATE-OBJECT`.

2.4.2 Host-Specific Object Names (HSONAMEs)

Every Prospero object has an <**hsoname**>. HSONAMEs are not guaranteed to be unique across time; if an object is deleted, a new object may be created that re-uses the old object's handle. However, at any particular moment, two Prospero objects on the same server are guaranteed to have unique hsonames, unless they are different versions of the same object (i.e., unless their VERSION-NUMBER fields are different.)

Two objects on different servers might have identical hsonames. In the current server implementation, the <**hsoname-type**> is always `ASCII` and the <**hsoname**> is almost always a full (i.e., starting with a slash (`/`)) UNIX filesystem pathname. See appendix C for more discussion of this. The <**hsoname-type**> token exists because some special filesystems may have non-`ASCII` <**hsoname**>s.

2.4.3 Versions

We intend to allow versioned objects. All objects, including directories, have a VERSION-NUMBER field. In the current server implementation, the VERSION-NUMBER field is always zero, which means "unversioned". It need not be in future server implementations.

In commands and responses, providing a zero value for the <**object-version**> token means to retrieve the current version of the object. Specifying explicit values means that a particular version of the object is required. Explicit object version numbers (when they are established) will always be positive integers.

2.4.4 Object `ID` fields

The `ID` field is a unique (or nearly unique) identifier for an object. If the underlying object changes (such as by editing), then some types of `ID`s will change and others will not. If two objects have the same `ID`, then they are considered by Prospero to be the same object. This is useful for distinguishing between two cases: (a) two links which happen to have the same <**name-component**> but different storage locations (a different host and handle pair), and which do not actually represent the same object, and (b) two links which have the same <**name-component**> and same `ID`, but possibly different storage locations — in case (b), the objects are replicas and the client should access whichever one it chooses to. (Code is currently under development to enable servers to return a list of replicas in order of nearness to the client.) Two links with the same <**name-component**> and no `ID` specification are considered by Prospero to be different objects, not replicas of one another.

A problem with the `ID` field is that there is not an accepted standard for universal document identifiers. An IETF working group has been formed to consider such identifiers, and the `ID` field exists in anticipation of their eventual agreement upon a standard. We expect that there will be more than one type of universal document identifier; therefore, there will be more than one <**id-type**>. Prospero supports objects with several possible `ID`s.[2]

For now, there are only two <**id-type**>s defined. The `REMOTE` <**id-type**> is used by the server when it has magic knowledge (perhaps through a local database) that two objects are the same. The `REMOTE` id type is a single element which is the printed decimal representation of a positive integer. This positive integer should be able to be stored in a 32 bit integer. A `REMOTE` id value of 0 means "unspecified". The `REMOTE` <**id-type**> cannot be sent across the network by clients

---

[2]This will go into the user's manual when we revise it. The user-level names for two links distinguished only by their `ID` fields are:

(**<name-component>##<id-type>**$_1$**#<id-value-token>**$_{1,1}$**#<id-value-token>**$_{1,2}$...**##<id-type>**$_2$**#<id-value-token>**$_{2,1}$**#<id-value-token>**$_{2,2}$...
or **<name-component>#<id-value-token>**$_1$**#<id-value-token**$_2$**>**...)

The second form matches any <**id-type**>. The first form is used to explicitly specify one or more <**id-type**>s.

in queries.[3] It is not guaranteed to be consistent across different queries to the server, but server implementors are encouraged to make the <**id-value**> token consistent across queries.

Many clients will locally generate `ID` fields to help the human user differentiate between conflicting links. These `ID` fields will have an <**id-type**> of `LOCAL` and an arbitrarily generated string as their <**id-value**>. They may not be sent over the network, and are not guaranteed to be unique from directory read to directory read, although implementors are encouraged to make them so. All available `ID` fields will be returned with every `LIST` operation.

[3] The <**id-value**> of the `REMOTE` type is the same as the <**magic-no**> token specified in version 1 of the Prospero protocol.

Chapter 3

Commands Reference

All Prospero commands are listed below.

3.1 VERSION

`VERSION` **<version-number>** [**<software-identifier>**]

This command requests or specifies the protocol version number. If the specified protocol version number is not supported, the response will be:

`VERSION-NOT-SUPPORTED TRY` **<min-version-number>-<max-version-number>**

or

`VERSION-NOT-SUPPORTED TRY` **<version-number>**

Where **<version-number>** or **<min-version-number>-<max-version-number>** are those versions that are supported. If no arguments are supplied (or if the argument is not a number), then the `VERSION` command will return the current protocol version number being used by the server in the form:

`VERSION` **<version-number>** [**<software-identifier>**]

If the **<software-identifier>** is specified, it identifies the software and version of the software that is generating the message.[1]

3.2 AUTHENTICATE

`AUTHENTICATE` **<options>** **<authenticator-type>** **<authentication-data>** [**<principal-name>**]*

[1] Software developers wishing to obtain software identifiers should send electronic mail to `pfs-administrator@isi.edu`.

This command authenticates the principal making the request. The **<authenticator-type>** is the type of the authenticator. It might be a password, a Kerberos authenticator, or data used by an alternative authentication mechanism. The currently supported values for **<authenticator-type>** are `UNAUTHENTICATED`, `KERBEROS`, and `HANDLE`.

If the **<authenticator-type>** is `UNAUTHENTICATED`, this is honored by the `ASRTHOST` ACL type. It is also honored by the `TRSTHOST` ACL type if the client is using a privileged port. If the **<authenticator-type>** is `UNAUTHENTICATED`, then the **<authentication-data>** should be the username of the user running the client. This username is be the principal referred to by the ACL.

If the **<authenticator-type>** is `KERBEROS`, then the **<authentication-data>** is a Kerberos authentication message authenticating the principal to the Prospero server. The ACL principal will be the same as the Kerberos principal. The ACL type will be `AUTHENT KERBEROS`.

If the **<authenticator-type>** is `HANDLE`, then the **<authentication-data>** is a handle returned in response to a previous `AUTHENTICATE` command.

The optional **<principal-name>**s are informational only for some authentication types, and exist only for human convenience. The server will extract the principal names from the **<authentication-data>**, but the names might be encrypted in the **<authentication-data>** or otherwise represented in a way that humans cannot easily decipher them. (For instance, this is the case with Kerberos version 5.) In the case of the `P_PASSWORD` authentication type, the **<principal-name>**s are not optional.

More than one `AUTHENTICATE` command may be sent in a single message. This can be used both to authenticate oneself as multiple simultaneous principals and to authenticate oneself using several methods.

The response may take one of several forms. If the authentication fails, then the response is:

`FAILURE AUTHENTICATION-DATA` [*explanatory text*]

One might get this response if an authentication handle has expired.

If it is computationally expensive for the server to validate the authentication data, it may want to cache the fact that the data has been validated, and return a handle that the client may use in future requests to the server:

`AUTHENTICATED` [**<authentication-handle>** [**<handle-expiration-time>**]]

The **<handle-expiration-time>**, if provided, is in ASN-TIME format.

The response may be another `AUTHENTICATE` command if the server needs to authenticate itself to the client.[2] The response may simply be:

`AUTHENTICATED`

to indicate that the authentication succeeded. If other commands are included in the same packet as the `AUTHENTICATE` request (this will almost always be the case), then successful execution of theose other commands implies that the authentication succeeded; in this case, the server is not required to include the `AUTHENTICATED` response.

Currently, no options are defined, so the **<options>** token is always the null string.

[2] XXX: Not currently implemented.

3.3 DIRECTORY

DIRECTORY <**hsoname-type**> <**hsoname**> [<**object-version**>][<**LF**>
SELECT
<**applies-to**> (FIELD *or* INTRINSIC *or* APPLICATION) <**attribute-name**>
<**attribute-value-type**>) [<**value-tuple-element**>]*]*

This command specifies the directory to be read or modified by the commands that follow as part of the same request. This command does not generate a response unless the selected directory is not a directory. In that case, the response:

```
FAILURE NOT-A-DIRECTORY
```

is returned, and the remainder of the request ignored.

3.4 ATOMIC

```
ATOMIC
```

This command specifies that all commands that follow are to be completed atomically. If one of the commands fails, then none of the commands are to have an effect. This may require undoing the effects of commands which have already completed. Note that this command works only for requests issued in the same message, and all commands must be executable on the single server to which the message was sent.

This command is not currently implemented.

3.5 LIST

LIST <**options**> COMPONENTS [<**name-component**>]*[<**LF**>
SELECT
<**applies-to**> (FIELD *or* INTRINSIC *or* APPLICATION) <**attribute-name**>
<**attribute-value-type**>) [<**value-tuple-element**>]*]*<**LF**>
FILTER ...

LIST is used to look up information stored in a directory. It must be preceded by a DIRECTORY command in the same request. Each optional <**name-component**> is a component of a multiple-component name relative to the directory specified in the DIRECTORY command. The last component may include wildcards. If no <**name-component**> is specified, the wildcard "*" is implied (i.e., all listable components in the directory will be listed).[3] If the result of the lookup of one component is another directory at the same storage site, then the directory server will repeat the process and look up the next component. When a lookup results in a dead-end, or a directory at another site,

[3]The protocol is currently in transition. The old format was as follows: Multiple component names are allowed, and are specified as a single token following the <**name-component**>. The multiple components within a single name are separated by the unquoted slash (/). Servers and clients for now are accepting both versions of the protocol message, which means that the 1st component of a multiple-component name must be quoted if it contains a slash.

the server will return an `UNRESOLVED` message, indicating which components remain to be resolved by the client.

If a space, tab, slash, or <**LF**> is to be included in a component of a single-component or multi-component name, the component must be quoted. Quoting a slash in a single-component or multi-component name means that the slash is not considered to separate components of a multi-component name, but rather to be a part of a single component. This somewhat awkward syntax allows us to have single components which contain slashes or horizontal whitespace or <**LF**>s.

3.5.1 Wildcards

The <**name**> tokens in the `LIST` command are the only places in the Prospero protocol where components may be specified with wildcards. There are two wildcard characters supported:

`*` Match zero or more characters.

`?` Match exactly one character.

Wildcards may only be used to expand at a single-component level; by this, we mean that the wildcarded <**name**> `foo?bar` would match the single-component name whose name in the current directory was `foo-bar`, but would not match the object whose multiple-component name in the current directory was `foo/bar`.

One may also wildcard a name by giving a full regular expression. To do this, the regular expression should be enclosed within parentheses. For instance, specifying (Anne.*) is equivalent to specifying the wildcarded name 'Anne*'.

In addition to any wildcards specified, a literal match for the wildcard will also happen.

3.5.2 Options

<**options**> contains a (possibly empty) list of options to the `LIST` command. If no options are specified, an empty list of options should be sent as a null token (`''`.)

Miscellaneous Options

Among the options supported are `EXPAND` which tells the remote directory server to expand any union entries in the directory. By default, the response will contain the names of union links to be included, and the client must submit a new query. A directory server can ignore the `EXPAND` option to the list command if it so desires. The `LEXPAND` option is the same as `EXPAND`, but indicates that the server should only expand union entries for local directories. The `LEXPAND` option is implied if a multiple component name is being resolved.

The `VERIFY` option tells the remote server that the purpose of the query is to verify whether the remote directory exists and is a directory. No components are to be returned; instead, the message returned on success is `NONE-FOUND`.

`ATTRIBUTES` option

The `ATTRIBUTES` option indicates that the attributes associated with the link are to be returned. (If the object is on the same host as the directory, then the server may ignore CACHED attributes and return OBJECT attributes instead.

ATTRIBUTES is optionally immediately followed by an argument list, which is a parenthesized +-separated list of attributes to be returned. If the ATTRIBUTES option is specified without any attributes requested, then it is just the same as if one had specified ATTRIBUTES(#INTERESTING). Note that union links do not normally have any interesting attributes attached to them, except for FILTER.

ATTRIBUTES(ID+FILTER) is always implied. For EXTERNAL links, ATTRIBUTES(ACCESS-METHOD) is always implied.[4]

There are some special arguments to the ATTRIBUTES option. It is not a good idea for application-defined attributes to have names that conflict with these special arguments; if there are such application-defined attributes, you may have to use the #ALL special argument to look at their values. More than one special argument may be specified, and one may mix special and non-special arguments. Also, note that it is never erroneous for a server to return more attributes than were asked for, nor is it erroneous for a server to return attribute information even if no ATTRIBUTES option was specified to the LIST command. A server that always behaved as if the ATTRIBUTES(#ALL) option were specified would be behaving legally, although it would not be terribly efficient.

#INTERESTING Return only interesting attributes. What constitutes an "interesting" attribute is server-dependent. This allows one to use Prospero for special applications.

#ALL If this argument is specified, and the object resides on the same host as the link, then the attributes stored with the object referenced by the link will be returned in addition to those stored with the link, and no cached attributes will be returned (they would be irrelevant). If the object resides on a host different from the link, then all attributes stored with the link are returned, but not those stored with the object referenced by the link.

#CACHED Return all attributes of type CACHED. If you specify #CACHED with #ALL, the server will return cached attributes in addition to those that would be returned if you'd just specified #ALL. It is not clear how useful this behavior is.

#OBJECT Return all attributes of type OBJECT. Note that this will not work if the object does not reside on the same host as the link.

#ADDITIONAL Return all ADDITIONAL attributes

#REPLACEMENT Return all REPLACEMENT attributes.

#LINK Return all LINK attributes.

#FIELD Return all fields (all system-defined standard attributes), except for ones that have already been returned as part of the LINK response, such as HOST, HANDLE, and DEST-EXP.

##FIELD Return all fields. (This option will be rarely used.)

#APPLICATION Return all application-defined attributes.

[4] This restriction seems odd, but it makes certain details of programming the client libraries much easier.

FILTER option

If the FILTER option is specified, the LIST command will be followed by one or more lines representing one or more PREDEFINED filters to be applied at the server.

The filters are applied in the order in which they follow the LIST command. Their format is just like that of FILTER information returned from the LIST command (see subsection 3.5.3), except that the <**execution-location**> field must always be SERVER, and the first four tokens ("ATTRIBUTE LINK FIELD FILTER") are omitted. The format is:

> FILTER <**filter-type**> SERVER <**pre-or-post**> PREDEFINED <**filter-name**> [ARGS [<**filter-arg**>]*]

The server will know that there are no more filter-specifying lines either when the end of the message is reached or it encounters a line that does not begin with the token FILTER.

3.5.3 Returned Information

On failure, a standard error response will be returned. On success, the response will be a sequence of entries containing information about the requested files.

Links

> LINK <**link-type**> <**target**> <**name-component**> <**host-type**> <**host-name**> <**hsoname-type**> <**hsoname**> <**object-version**> [DEST-EXP <**dest-expiration**>]

In the case of links to objects, <**dest-expiration**> is the value of the object's DEST-EXP field.

The <**link-type**> token is defined as follows:

L Normal link (Symbolic link, link to a Prospero object, or link to an external object)

U Union link to a directory

The <**target**> token is defined as follows: (Also see the discussion of this in section A.2.2. A link may be a link to an object. In this case, the value of the <**target**> token will be the same as the value of the object's BASE-TYPE attribute[5] (that is, OBJECT, FILE, DIRECTORY, or DIRECTORY+FILE.) Other values for <**target**> are:

SYMBOLIC Symbolic link. The <**host-type**> token will be VIRTUAL-SYSTEM, the <**host-name**> token will be the name of the virtual system being linked to (specified as a path within the current virtual system — we usually use the conventional ugly-name of the virtual system), the <**hsoname-type**> will be ASCII, and the <**hsoname**> will be the full pathname of the object being linked to within the virtual system.

EXTERNAL This is a link to an object which is not stored on a host running Prospero. Unlike other types of links, EXTERNAL links have an ACCESS-METHOD [6] attribute which is returned as a cached attribute. This is because EXTERNAL links cannot have any object attributes. Therefore, if one wishes to add attributes to an EXTERNAL link that would normally be object attributes, one must specify them as CACHED, ADDITIONAL, or REPLACEMENT attributes.

[5] For an explanation, see section 2.4.1.

[6] See section A.1.1 for a description of the ACCESS-METHOD field.

If the principal requesting the listing has no read access to a link, all fields in the link except for **<name-component>** will be returned with the token `NULL`, which is not otherwise valid in a returned link.

Link Attributes

If attributes were requested, the link will be followed by lines that specify the values of the requested attributes. The form of the response will depend on whether the attribute is associated with the link, or with the actual object. If the attribute is associated with the link, the **<applies-to>** token indicates whether it applies to the `LINK` or the object, and if the object whether it is a `CACHED` attribute, a `REPLACEMENT` attribute, or an `ADDITIONAL` attribute. In case of conflicts between attributes associated with the link and those associated with the object, a cached attribute is superseded, a replacement attribute takes precedence, and an additional attributes leaves both intact.

Attributes may sometimes be returned even if they were not explicitly requested. As a case of this, the ACCESS-METHOD attribute is always returned for `EXTERNAL` links.

(`ATTRIBUTE` **<applies-to>** `FIELD` **<attribute-name>** [**<value-tuple-element>**]$^+$
or
`ATTRIBUTE` **<applies-to>** (`APPLICATION` *or* `INTRINSIC`) **<attribute-name>** **<attribute-value-type>** [**<value-tuple-element>**]$^+$)

[7]

Attributes may have multiple values, in which case one `ATTRIBUTE` line is returned for each value. In the case of multiple values, the order in which attributes is returned may be significant to the application. The Prospero server will maintain the order of attributes.

<attribute-value-type>s are `SEQUENCE`, `LINK`, and `FILTER`. `SEQUENCE` is the most general **<attribute-value-type>**. It is a sequence of zero or more ASCII strings, and all other attribute value types are subtypes of it. Since application attributes are, by definition, application-specific, we are not constraining the form of a sequence. However, if an application wishes to use application-specific subtypes of `SEQUENCE`, we encourage application authors to use the first element of the attribute's value as a keyword describing the particular subtype of `SEQUENCE`.

One possible **<attribute-value-type>** is `LINK`. The **<value-tuple-element>** tokens returned in this case are the same as the `LINK` response to the `LIST` command. In that case, the **<name-component>** token will be ignored, and will usually be a zero-length string (`''`).) The return format would be:

`ATTRIBUTE` **<applies-to>** `APPLICATION` **<attribute-name>** `LINK` (`L` *or* `U`) **<target>** **<name-component>** **<host-type>** **<host-name>** **<hsoname-type>** **<hsoname>** **<object-version>** [`DEST-EXP` **<dest-expiration>**]

[7] **CHANGE IN FORMAT**: It used to be the case that the line returned for FIELD attributes did not include an **<attribute-type>** token. This has been changed. The older format will continue to work until all Prospero applications before Alpha.5.1 are gone.

If the value of an attribute is a link, the link might itself have attributes. When such sub-attributes are returned, the word `ATTRIBUTE` is immediately followed by `>` signs to the appropriate nexting level. If a link attribute has sub-attributes, they will all be sent with the link.

All available `ID` fields will always be returned with every `LIST` operation, using the `ID` return format.[8]

Filters

User-defined attributes and the field FILTER may have the **<attribute-value-type>** of `FILTER`.[9] If a link has one or more filters attached to it, they are specified as one or more instances of the link's FILTER field. One `ATTRIBUTE` line will always be returned for each filter. The lines are returned in the same order in which the filters will be applied to the link. The return format is:

> `ATTRIBUTE`[`>`]*`LINK FIELD FILTER` **<filter-type>** **<execution-location>** **<pre-or-post>**
> (`PREDEFINED` **<filter-name>**
> *or* `LINK L` **<target>** **<name-component>** **<host-type>** **<host-name>** **<hsoname-type>** **<hsoname>** **<object-version>** [`DEST-EXP` **<dest-expiration>**]) [`ID` **<id>** **<info>**]*
> [`ARGS` [**<filter-arg>**]*]

The values for **<filter-type>** are: `DIRECTORY`, filter is applied to the current directory; `HIERARCHY`, filter is applied to all directories reachable through the filtered link; `OBJECT` filter is applied to an object other than a directory; `UPDATE`, filter is applied when updating the directory.

<execution-location> refers to where the filter will be executed. Filters are usually executed on the `CLIENT`, but may be executed on the `SERVER`.

<pre-or-post> is `PRE` if the filter is to be applied before union links are expanded and `POST` if the filter is to be applied after union links are expanded. `POST` is the common case for client filters. Server filters must be `PRE` at this time, since the server does not yet perform remote expansion of union links.

Filters may be `PREDEFINED`, which means that it is expected that the appropriate client or server will understand the predefined name, or they may be `LOADABLE`, which means that they are object code that will be dynamically linked with the client or server.[10] There may also be server-specific `PREDEFINED` filters for special applications (such as Archie). Their **<execution-location>** will always be `SERVER`.

One may specify zero or more arguments to the filter, following the `ARGS` keyword.

Unresolved Components

An `UNRESOLVED` response lists the components of the name that must be further resolved relative to the immediately preceding `LINK` response or responses, and is used when not all components of

---

[8] Please note again that ID fields have not been fully implemented, in the absence of an appropriate standard; comments about them in this document are subject to change.

[9] XXX: This should go into the attribute documentation

[10] The security issues with loadable filters have not yet been fully addressed. Partly because of this, right now, all implementations except for the VAX implementation will only support `PREDEFINED` filters, and even the VAX implementation is not configured to support loadable filters unless you specially compile it to do so.

a multiple-component name were resolved.

`UNRESOLVED` <**components**>

The text of the <**components**> must be a proper suffix of multiple-component name sent across as an argument to the `LIST` command. For instance, to the command

`LIST '' COMPONENTS ulink-test2/murali/ROOT`

the only two legal `UNRESOLVED` responses are

`UNRESOLVED murali/ROOT UNRESOLVED ROOT`

No files found

If no files are found, the reply will be:

`NONE-FOUND`

3.6 LIST-ACL

`LIST-ACL` <**options**> [<**name-component**>][<**LF**>
`SELECT`
<**applies-to**> (`FIELD` *or* `INTRINSIC` *or* `APPLICATION`) <**attribute-name**>
<**attribute-value-type**>) [<**value-tuple-element**>]*]*

This request is used to list the access control list for the directory specified in the previous `DIRECTORY` command, or for a link within the directory. The optional component is required only if requesting the `ACL` for a link within the directory (option = `LINK`). It is to be left out when requesting the `ACL` for the directory itself (option = `DIRECTORY`).

The response will be zero or more lines of the form:

`ACL` <**entry-type**> [<**authentication-type**> [<**rights**> [<**principal-name**>]*]]

If no `ACL` entries are listed, then the response is `SUCCESS`. Fields inappropriate to a particular <**entry-type**> need not be sent, but if they are sent, they should be sent as a zero-length string. For instance, the `DEFAULT`, `SYSTEM`, and `DIRECTORY` ACL types do not use the <**authentication-type**>, <**rights**>, and <**principal-name**> fields.

If the ACL's permissions state that it cannot be viewed (`v` or `V` permission), then the response is

`FAILURE NOT-AUTHORIZED` [*optional multi-token explanatory text.*]

Possible values for <**entry-type**> include `NONE` (allows access to no principals; in other words, it's a no-op), `ANY` (grants access to all principals), `OWNER` (grants access to the principal specified by the link or object's OWNER field; i.e., the one who owns the link or object), `NAMED`, `GROUP` (In this case, the <**authentication-type**> token represents the group certification method), `DIRECTORY`, `DEFAULT`, `SYSTEM`, `AUTHENT` (means that an authentication method is used; the <**authentication-type**> token specifies the authentication method. The only currently supported value for <**authentication-type**> is `KERBEROS`.), `ASRTHOST`, and `TRSTHOST`. An example:

```
ACL ANY '' rlvY
ACL ASRTHOST '' ALRMDI prospero
ACL AUTHENT KERBEROS ALRMDI swa@ISI.EDU bcn@ISI.EDU
ACL DEFAULT '' ''
ACL SYSTEM '' ''
```

We interpret a null link ACL as the `DIRECTORY` ACL. We interpret a null directory ACL as the `DEFAULT` ACL.

See the Prospero User's Manual for a discussion of the order of evaluation of ACLs.

If we want to find out the value of a `NAMED` ACL (named ACLs are shorthand for longer lists, and are local to the server) (`XXX` this should go into the user's manual), then we use the `NAMED` option, and the **<name-component>** is replaced with the (possibly quoted) name of the named ACL. Note that named ACLs are not currently supported.

If we want to find out the value of an included ACL (Currently, the only included ACLs are `SYSTEM`, `DEFAULT`, and `OVERRIDE`. `DIRECTORY` is also an included ACL for the purposes of the protocol, but the value of the `DIRECTORY` ACL will change from directory to directory, and therefore it is not a server-wide stable name.) [11], then we use the `INCLUDED` option, and the **<name-component>** is replaced with the name of the included ACL. Note that retrieval of included ACLs is not currently supported.

If we want the ACL for a filesystem object (instead of for a link or directory), then we use the `OBJECT` option, and the **<name-component>** is replaced with **<hsoname-type>** **<hsoname>** [**<object-version>**]. Object ACLs are not currently implemented, because they wouldn't do anything useful — we currently do not have an access method which understands Prospero ACLs.

Within the Prospero protocol, the quoting mechanism works on principal names, and works recursively for distinguishing components of principal names. (This is currently only used by Kerberos, but may also be needed by other authentication mechanisms.)

See the Prospero User's manual for a further discussion of ACLs.

3.7 GET-OBJECT-INFO

```
GET-OBJECT-INFO <requested-attributes> <hsoname-type> <hsoname>
   <object-version>[<LF>
SELECT
   <applies-to> ( FIELD or INTRINSIC or APPLICATION ) <attribute-name>
   <attribute-value-type> ) [<value-tuple-element>]* ]*
```

This command requests information about an object. **<requested-attributes>** is a list of those attributes that are desired. Multiple attributes may be separated by "`+`" signs (with no intervening white space), just as options lists are separated.

Special attribute names may be specified, just as they may be for the `LIST` command. The special names are: `#OBJECT` (synonymous with `#ALL`), `#FIELD`, `#INTERESTING`, `#ALL`, and `#APPLICATION`.

The response can be multiple lines, each containing a value for an attribute. The response will be the same as for the `ATTRIBUTE` option to the `LIST` command. However, the **<applies-to>** field will always be `OBJECT`

[11] XXX: this information will go into the user's manual when we revise it

An object that has migrated to another host may have its FORWARDING-POINTER attribute queried with `GET-OBJECT-INFO`, but attempts to retrieve any other attributes will return a `FORWARDED` message.

If no matching attributes for the object were found, the response is `NONE-FOUND`.

3.8 EDIT-OBJECT-INFO

`EDIT-OBJECT-INFO` <**modification-request**> <**hsoname-type**> <**hsoname**>

This command is used to change the attributes of a Prospero object. It is followed by an arbitrary (zero or more) number of `ATTRIBUTE` specification lines. See the `LIST` command for a definition of these lines. The <**applies-to**> will always be `OBJECT`.

In the command, one may request to add a new instance of an attribute, delete an instance, delete all instances, or replace all instances. The <**modification-request**> will be `ADD`, `DELETE`, `DELETE-ALL`, or `REPLACE`. If you want to do more than one thing to an object in the same request (e.g., if you want to add one attribute instance and replace another), and you want to avoid letting the object exist in an intermediate and possibly inconsistent state, then you can send two `EDIT-OBJECT-INFO` commands in the same message, along with the `ATOMIC` command.

If you are deleting all instances of an attribute, then the <**attribute-value**> you specify in your `ATTRIBUTE` specification lines will be ignored, and will usually just be the null string. `DELETE-ALL` just checks for matches on the attribute name and the attribute namespace (`APPLICATION`, `FIELD`, or `INTRINSIC`).

`DELETE`, on the other hand, checks for an exact match on the attribute name, namespace, type, and value. (The current implementation does not check for sub-attributes of a value of type link.

`REPLACE` may be specified even if the attribute does not yet have any instances specified for it.

One may use `EDIT-OBJECT-INFO` to edit the BASE-TYPE attribute to remove the `FILE` type from it only if the file is empty (zero length), and one may remove the `DIRECTORY` type from it only if the directory is empty (contains no links).

3.9 CREATE-LINK

`CREATE-LINK` <**options**> (`L` *or* `U`) <**name-component**>
<**target**> <**host-type**> <**host-name**> <**hsoname-type**> <**hsoname**>
<**object-version**> [`ID` <**id-type**> <**id-value**>]

This command creates a new link in the current directory. The <**name-component**> may not be null, even if the link is a union link. If filters or other information must be added, the `EDIT-LINK-INFO` command should be used once the link has been created.

The <**options**> may include `REPLICA` to add a link to a directory that already contains a link with the same <**name-component**>. `REPLICA` indicates that the new link and the existing link or links with which it conflicts are replicas. Normally, if one attempts to add a link with a conflicting <**name-component**>, the response is:

```
FAILURE ALREADY-EXISTS
```

if the new link duplicates an existing link (if all of the arguments to it are the same as those of an existing link), and

FAILURE NAME-CONFLICT

if the new link has information which conflicts with an existing link.

For directories whose DIRECTORY-ORDERING is UNSORTED, one of these three additional options may be specified:

BEFORE The CREATE-LINK command is immediately followed by a LINK line, describing the link before which the new link is to be inserted in the directory listing.

AFTER Same as above, but the new link is inserted after the specified link rather than before it.

LAST This is the default. The new link will be the last item in the directory.

3.10 DELETE-LINK

DELETE-LINK <**options**> <**name-component**> [<**LF**>
SELECT
<**applies-to**> (FIELD *or* INTRINSIC *or* APPLICATION) <**attribute-name**>
<**attribute-value-type**>) [<**value-tuple-element**>]*]*

This command is used to remove an entry from a directory. The options are currently blank.

If the SELECT lines are not specified and there are multiple links with the same <**name-component**>, the first one matching will be deleted.

3.11 EDIT-LINK-INFO

EDIT-LINK-INFO <**modification-request**> <**name-component**> [ID <**id-type**> <**id-value**>]

This command modifies information associated with a link. The interpretation of the <**modification-request**> and of subsequent lines is exactly the same as in the EDIT-OBJECT-INFO command.

3.12 EDIT-ACL

EDIT-ACL (LINK+<**options**> <**name-component**>
or OBJECT+<**options**> <**hsoname-type**> <**hsoname**>
or DIRECTORY+<**options**>)
[ID <**id-type**> <**id-value**>]<**LF**>
ACL <**entry-type**> <**authentication-type**> <**rights**> <**principal**>

This command is used to modify an access control list for a directory, for a link within a directory, or for an object. If modifying the ACL for a directory or for a link within a directory, the directory is specified in the previous `DIRECTORY` command. Another option indicates the operations to be performed (one of `ADD`, `SUBTRACT`, `INSERT`, `DELETE`, `SET` or `DEFAULT`), and whether to override the automatic inclusion of the system ACL (`NOSYSTEM`), or limited administer access for the client (`NOSELF`).

The server will return `SUCCESS`, `FAILURE`, or `FORWARDED`, as appropriate.

3.13 CREATE-OBJECT

(`CREATE-OBJECT` `PHYSICAL+`<**options**> [<**hsoname-type**> <**hsoname**>]
or `CREATE-OBJECT` `ADD+`<**options**> [<**hsoname-type**> <**hsoname**>]
or `CREATE-OBJECT` `VIRTUAL+`<**options**> <**name-component**>) <**LF**>
[`ATTRIBUTE` <**attribute-specifying-tokens**><**LF**>]*
[`ACL` <**ACL-specifying-tokens**><**LF**>]*

This command will create an object with the specified <**name-component**> and will return the identifier that can be used to open it. The <**options**> may include any or all of `DIRECTORY`, `FILE`, and `OBJECT`. If the `FILE` option is specified, the new file is created empty. If the `DIRECTORY` option is specified, the new directory is created empty (i.e., not containing any links.).

The object will be created with whatever attributes present that are specified in the `ATTRIBUTE` lines.

If <**options**> includes `VIRTUAL`, then <**name-component**> is a name relative to the current directory, and a link to the new object is added to the current directory.

If the `ADD` option is specified, then one is adding a base type to an already created object. Exactly one of the `ADD`, `VIRTUAL`, and `PHYSICAL` options must be specified.

If the `PHYSICAL` option and a <**hsoname-type**> <**hsoname**> pair are specified, then the server will attempt to create an object with that <**hsoname-type**> <**handle-pair**>. If the `PHYSICAL` option without a following pair is specified then the server will choose a free <**hsoname-type**> <**hsoname**> pair.

Upon success, the server will return:

`CREATED` <**hsoname-type**> <**hsoname**>

It is up to the application to create a link to the new object.

3.13.1 Specifying Attributes

The new object's fields will be set to system defaults, but specifying a following series of `ATTRIBUTE` descriptions allows the creator of a file to override those defaults, and to specify any application-defined attributes. The form for each `ATTRIBUTE` line is the same as that returned by the `LIST` command.

3.13.2 ACLs

If creating an object with the `VIRTUAL` option, the access control list for the new object will initially be a copy of the access control list for its containing directory. If creating an object with the

PHYSICAL option, the access control list for the new object will contain the DEFAULT ACL and the SYSTEM ACL. In addition, for both options, one or more entries will be automatically added to the ACL granting its creator all rights. The entry authentication type or types will be appropriate for whatever the user used to authenticate himself or herself. (Either AUTHENT KERBEROS or ASRTHOST or TRSTHOST.)

If the LPRIV option is specified for a directory, instead of this entry, only those rights needed to allow the creator to set up the directory (list, read, insert, and administer) will be added, and (if VIRTUAL was specified), only if the creator does not already have such rights through the ACL that was included from the parent directory.[12]

If VIRTUAL was specified, the ACL for the new link will by default be empty, which means that the default rights for the directory will apply to the link.

After the ACL entries mentioned above are installed, the ACL entries specified as part of the CREATE-OBJECT command will be added to the front of the list.

3.13.3 Some Error Conditions

If the user attempted to set a field whose name the server does not recognize, or to set a application-defined attribute to a type that the server does not recognize, or to set the value of any attribute with a string that the server cannot convert into the data type the server expects (e.g., too few tokens when setting an attribute of type LINK), or to set an unrecognized ACL type, the response is:

ERROR *explanatory text*

If the user specified an explicit <**hsoname-type**> and <**hsoname**> with the PHYSICAL option, but they were already in use, or if the user specified a component with VIRTUAL, but the component is already in use, the error message is:

FAILURE NAME-CONFLICT [*explanatory text*]

3.13.4 No DELETE-OBJECT command

Although there is a DELETE-LINK command, there is *no* DELETE-OBJECT command. That's because Prospero objects will have their storage reclaimed when their TTL expires (when no link to them is refreshed before their expiration date). At the moment, this storage reclamation feature is not implemented.) [13]

3.14 UPDATE

UPDATE <**options**> COMPONENTS [<**name-component**>]*

This command tells the server to check each of the named components in the current directory for forwarding pointers. If the referenced object has moved, the link will be updated. The server

[12] If you really want to shoot yourself in the foot, you can then call EDIT-ACL to remove these few rights. We will let you do this, but we are not making it easy for you.

[13] XXX: This feature must be specified. In addition, we must specify methods for refreshing links, garbage collection, and notification of the impending demise of objects.

will send Prospero protocol messages across the network to the targets of links to Prospero objects in order to check whether the target of a link has moved. (External links and symbolic links will not be checked.) If no components were specified, all components in the directory will be checked. Each <**name-component**> may contain wild cards.

On success, the `UPDATE` command returns the number of links that were modified.

`UPDATED` <**number**> `LINKS`

On failure, the `UPDATE` command returns the number of links it was unable to update.[14]

`FAILED TO UPDATE` <**number**> `LINKS`

Wildcards

3.15 STATUS

This command requests the current status of the server. A humanly readable multiple line response is returned. The response may be presented to the user without additional processing. The response must conform to the following requirements so that it may be read by a program if desired.

The first line must include the server's software version identification enclosed in parenthesis, and the host name of the server. The name of the host should be the name that appears on local links generated by the server; it might not be the primary name of the host. The version identifier must be the first string that appears in parenthesis, and the host name must be the string that immediately follows the version identifier.

If a line contains a colon (:), the string preceding the colon identifies the meaning of the text that follows the colon. Reserved identifiers include `Contact`, `Started`, `Memory`, `Data`, `Root`, `AFTP`, `AFS`, and `DB`. The identifiers are case insensitive. If present, `AFTP` identifies the part of the file system accessible by anonymous `FTP`.

More than one `DB` line may be present. A `DB` line may contain more than one item on it; if it does, the items must be separated by spaces. Each item on a `DB` line is an initial prefix which this particular server recognizes to mean that, if this server receives a system-level name with this prefix followed by a slash, the remaining contents of the line are fed to a database program which translates it into a reference to a virtual directory.

Other than for the first line of the response, implementations are free to add or modify lines that do not contain a colons. A sample status response follows:

```
Prospero server (Beta.4.2B) JUNE.CS.WASHINGTON.EDU
Requests since startup 4096 (3609+377+2 67+29+0 9 0+0+0 0 3) OF
Started: 26-Aug-91 15:14:56 UTC
Contact: bcn@cs.washington.edu
 Memory: 0(118)vl 0(4)at 0(5)acl 0(1)fi 1(1)pr 2(2)pt 0(711)str
   Data: /u1/vfs/pfsdat
   Root: /
     DB: ARCHIE GOPHER(70) GOPHERGW WAIS
   AFTP: /homes/june/ftp
```

[14] This error message may change in response to future needs.

Since the responses to this command are so free in form, it is unlikely you would want to send additional requests to the Prospero server along with the `STATUS` request, since it would not be easy for a program to separate the replies to them from the free-form text.

Chapter 4

Standard Responses

4.1 SUCCESS

`SUCCESS` [*identifying-info*]

A command that does not generate any output returns this response if successful.

4.2 FORWARDED

`FORWARDED` **<host-type>** **<host-name>** **<hsoname-type>** **<hsoname>**
<version> [`DEST-EXP` **<dest-expiration>**]
[`ID` **<id-type>** **<id-value>**]*

This response is returned when the object that is the target of an operation has moved. The client can retry the response using the corrected information.

4.3 ERROR

`ERROR` *text*

This response is returned to indicate an error encountered when parsing the request. The error might be a protocol error, or it might be the result of the server's inability to recognize a keyword or data type. *text* describes the error.

4.4 FAILURE

`FAILURE` **<identifying-info>** [*optional text*]

This response is returned when an operation can not be performed. The defined values for **<identifying-info>** appear below. If present, the *optional text* provides additional information about the failure.

```
NOT-FOUND ( FILE or ACL or OBJECT or FILTER or PARAMETER or ATTRIBUTE )
ALREADY-EXISTS
NAME-CONFLICT
AUTHENTICATION-REQUIRED
NOT-A-DIRECTORY
NOT-AUTHORIZED
AUTHENTICATION-DATA
SERVER-FAILED[1]
AMBIGUOUS
BAD-VALUE
FILTER-APPLICATION
```

4.5 WARNING

`WARNING` <**identifying-info**> [*optional text*]

This response is returned to indicate a warning condition which does not affect the correctness of the response. This message can be used to indicate that the client is using an old version of the protocol that, while supported, should be phased out. It can also be used to inform the client of future changes on the server or scheduled downtime. The defined values for <**identifying-info**> appear below. *optional text* is optional text that provides additional information about the warning.

`OUT-OF-DATE`
`MESSAGE`
Any <**identifying-info**> *that can follow a* `FAILURE` *message*

Prospero clients are strongly encouraged to present warnings to the user.

[1]This is used in the following situations, among others: if an internal table in the server fills up, or if the server cannot allocate enough memory to handle the request, or if the server detects an internal inconsistency in its database, or if the server has not implemented a command specified in the protocol.

Appendix A

Directory, Link, and Object Attributes

This appendix describes the object and directory information maintained by the Prospero file system.

Please note that this appendix is in very rough form. Many of the attribute definitions have not yet been properly written. A lot of the attribute documentation can be found in section 3.5.3 of this document.

Attributes have three namespaces associated with them: `APPLICATION` attributes, `INTRINSIC` attributes, and `FIELD`s. Attributes in the `FIELD` namespace have a registered meaning, and are understood by all clients and servers that use them. `APPICATION` attributes are defined by users and application programs and are unrestricted. Intrinsic attributes have special registered meanings, providing prossibly transient or derived information. The server has flexibility about whether it allows you to change these things. Modifying them may have special implications — for instance, setting the SIZE of a file to "`0 bytes`" will truncate the file. `INTRINSIC` attributes gare either not modifiable, or else the server uses special mechanisms to modify them.

In addition, there is a fourth namespace used internally by routines running on the Prospero server. `INTERNAL` attributes are never sent across the network; they cannot be read with GET-OBJECT-INFO or LIST and cannot be modified with EDIT-OBJECT-INFO. However, they do appear in the server's data files.

A.1 Objects

A.1.1 Attributes

Valid attributes include ACCESS-METHOD, CLOSURE, DESCRIPTION, FORWARDING-POINTER, KEYWORDS, LAST-REFERENCED, LAST-WRITER, LOCKS, OWNER, REPLICAS, SIZE, STORAGE-LOCATION, TTL, TTL-EXPIRES. VERSION-NUMBER, VIRT-SYS, WELL-KNOWN-NAMES, and WRITE-DATE.

Prospero maintains the following system defined attributes for each Prospero object:

BASE-TYPE See section 2.4.1 for a description of this attribute.

[0]**RATIONALE**: Under the version 1 protocol, the means of obtaining the access method information for `EXTERNAL`

ACCESS-METHOD [1] This is a tuple of at least 5 elements. The first element is the name of the access method. Currently supported values are `AFTP`, `GOPHER`, `AFS`, `NFS`, `FTP`, `WAIS`, and `LOCAL`. The next 2 elements are the <**host-type**> and <**host-name**> of the host to which the access method should connect in order to retrieve the object. A port number may be included as part of the hostname, if relevant. If they are zero-length tokens (`''`), then they default to the <**host-type**> and <**host-name**> specified in the link. The next 2 elements are the <**hsoname-type**> and <**hsoname**> which the access method will use to retrieve the object. If they are zero-length tokens, then they default to the <**hsoname-type**> and <**hsoname**> specified in the link. This constraint on format applies to all access methods. If a particular type of access method doesn't need one of these fields, then it still must be specified, but the access method is free to ignore it. The whole point of the `''` shorthand is merely to save bytes in the protocol messages. It is always appropriate to send them fully expanded, and not use the `''` shorthand.

The sixth and subsequent elements are dependent upon the particular access method. For `AFTP` and `FTP`, the sixth element will be `BINARY` or `TEXT`. For `NFS`, the sixth element will be the name of the filesystem on the remote host. `LOCAL`, and `AFS` have only five-element names.

The `GOPHER` access method may have five or six elements in the name. The actual protocol used to retrieve a document or object through Gopher varies depending on its Gopher item-type. (See Internet RFC 1436 for details on the interpretation of the Gopher item-type characters.). This item type is usually the 1st character of a Gopher document or object's selector string (the <**hsoname**> element of the access method). However, the protocol does not require that this be the case. If a sixth token is present in a `GOPHER` access method, it will be treated as the Gopher item-type character; otherwise, the item-type will be taken from the 1st character of the <**hsoname**> element.

Some examples:

```
ATTRIBUTE FIELD ACCESS-METHOD GOPHER
   INTERNET-D MERMAID.MICRO.UMN.EDU(150)
   ASCII '1/Fun Stuff/Pyrotechnics/PyroGuide 1'[2]
```

This access method could also be displayed in the six-token format as:

```
ATTRIBUTE FIELD ACCESS-METHOD GOPHER
   INTERNET-D MERMAID.MICRO.UMN.EDU(150)
```

and `FILE` links was different; this made the code more complicated than it needed to be.

The reader may well ask why the host should be specified in the value of the ACCESS-METHOD field. Isn't the host name already specified in the HOST field? Including the host name as part of the attribute's value has some advantages:

- The value of the ACCESS-METHOD field is all one needs in order to retrieve the file.
- One might have a Prospero server running on only one host in a cluster, where the objects themselves are actually stored on another fileserver. Then, one would want to be able to have the ACCESS-METHOD host be different from the host which is listed as the one having the final word on the status of the object.
- This system also allows us to have a gateway server, which is able to translate directory formats from other distributed file systems (e.g., gopher), but directs the client to retrieve the files themselves from a host that is not the gateway server.

[2] This file recently disappeared from the server. If you have a copy, please send it to `swa@isi.edu`.

```
ASCII '1/Fun Stuff/Pyrotechnics/PyroGuide 1' 1
```
[3]

Andrew File System names are the same irrespective of what host one is using them from. Therefore, the <**host-name**> token in the access method is irrelevant and is ignored by the client.

```
ATTRIBUTE FIELD ACCESS-METHOD AFS DUMMY DUMMY
   ASCII /grand.central.org/doc/afs/dce/usenix90/README
```

This file can be retrieved by NFS mounting the `/auto/gum/gum` filesystem on the host PROSPERO.ISI.EDU. Note that the server is responsible for knowing which client machines it will allow to NFS mount that filesystem.

```
ATTRIBUTE FIELD ACCESS-METHOD NFS INTERNET-D PROSPERO.ISI.EDU
   ASCII ftp/pub/prospero/README-prospero-documents /auto/gum/gum
```

<**hsoname**> tokens in the `FTP` access method are full local hostname paths that one would give when using full FTP to the host.

```
ATTRIBUTE FIELD ACCESS-METHOD FTP INTERNET-D PROSPERO.ISI.EDU
   ASCII /ftp/pub/prospero/README-prospero-documents TEXT
```

<**hsoname**> tokens in the `AFTP` access method are pathnames relative to the root of the anonymous FTP area.

```
ATTRIBUTE FIELD ACCESS-METHOD AFTP INTERNET-D PROSPERO.ISI.EDU
   ASCII /pub/prospero/README-prospero-documents TEXT
```

If one is querying from a host which has a file accessible via the local filesystem, and if the server has knowledge of this fact, a `LOCAL` access method will also be returned for a file. For the `LOCAL` access method, the <**host-name**> token in the access method is ignored.

```
ATTRIBUTE FIELD ACCESS-METHOD LOCAL '' ''
   ASCII /auto/gum/gum/ftp/pub/prospero/README-prospero-documents
```

CLOSURE The <**attribute-type**> of thie attribute is `LINK`. It is a link to the virtual system to be used when resolving names embedded in the object. The link's <**name-component**> will be ignored, but by convention it is the ugly-name of the virtual system.

OWNING-VIRTUAL-SYSTEM The <**attribute-type**> of this attribute is `LINK`. It is a link to the description of the virtual system which is considered to own the object. (Every virtual system, by convention, has a link to its description named `/VS-DESCRIPTION`. For directories, if you are logged into the directory's OWNING-VIRTUAL-SYSTEM , then if you try to change the directory, your client will assume that you really want to try to change the directory. If you are logged into a different virtual system from the directory's OWNING-VIRTUAL-SYSTEM , then if you try to change the directory, your client will assume that you really want to customize your own virtual system, but not affect the master directory. Your intentions are completely distinct from whether you actually have write permission on the directory. If you try to change a directory that you can't write to (if its OWNING-VIRTUAL-SYSTEM is the same as the virtual system you're logged into, but you don't have permission to write to it), then the current clients will all give you a "permission denied" error message, and the nice ones will suggest that you might want to change virtual systems and customize instead. The

[3] This file recently disappeared from the server. If you have a copy, please send it to `swa@isi.edu`.

link's <**name-component**> will be ignored, but by convention it is the ugly-name of the virtual system.

VERSION-NUMBER The <**attribute-type**> of this attribute is `SEQUENCE`. It is represented in the protocol as a decimal number. It is currently always zero; see section 2.4.3 for further details.

OWNER The principal responsible for the object. The <**attribute-type**> of this attribute is `SEQUENCE`. It is a tuple of three or more elements. The first element is an ACL <**entry-type**> and the second element is an ACL <**authentication-type**>. The third and subsequent elements are <**principal-name**>s. The first two elements are needed because they indicate the namespace in which the <**principal-name**>s are to be resolved. There may be multiple instances of the OWNER field, if the owner wants to be registered according to several authentication systems. For instance, one of the authors of this document uses an OWNER field that looks like this:

```
ATTRIBUTE FIELD OWNER ASRTHOST '' swa@128.9.*.*
ATTRIBUTE FIELD OWNER AUTHENT KERBEROS swa@ISI.EDU swa/padmin@ISI.EDU
```

Although the OWNER field may be referred to with the `OWNER` ACL type, we don't really encourage people to use it as a shorthand for granting access rights. Its primary purpose is informational.

FORWARDING-POINTER This attribute has type `LINK`. The link's <**target**> is `FP`. It is set for objects that have migrated to another host. The target of its value is the new location of the object. If an object has moved to a new host, its FORWARDING-POINTER will be returned in a `FORWARD` protocol message in response to any attempts to access it.

Last writer, write date, etc.

Version info (optional). Number of versions to keep, version number, etc.

Attributes or keywords (optional). User specified.

Short description (optional).

Locks (optional).

List of replicas (optional).

Replication type (optional).

Other replication information (optional).

Time to live . The lifetime for newly created or refreshed links to the object.

TTL-EXPIRES This is the TTL plus the time that a link to the object was last created or refreshed.

Back links . A possibly incomplete list of directories with links to the object.

Note that the object's name is not one of its attributes. The object's name is the concatenation of the name components starting from the active virtual system. An object may have different names in different virtual systems, or even multiple names within a single virtual system[4].

A.1.2 Objects stored on UNIX

Objects stored on UNIX servers have the following attributes defined for them. These are all `INTRINSIC`:

SIZE A single-element `SEQUENCE` specifying the number of bytes used to store the object. Its format is "<**number**> `bytes`". Note the embedded space.

NATIVE-OWNER A single-element `SEQUENCE`. The UNIX username on the server of the user who owns this file.

NATIVE-OWNER A single-element `SEQUENCE`. The UNIX group name on the server of the group associated with this file.

LAST-MODIFIED A single-element `SEQUENCE`. The ASN-TIME representation of the last modified time of this object.

UNIX-MODES A single element `SEQUENCE` consisting of a ten character string. The first character is "`l`" for symbolic links, and "`-`" otherwise. The remaining 9 characters are the user, group, and other protection bits in the standard format returned by "`ls -l`".

A.1.3 Persistence

An object continues to exist until the last non-expired link referencing the object is removed. If a user wishes to recover the storage space for an object, it is flagged for removal. When links to the object are refreshed, notification is sent that the object is about to disappear, and anyone wanting to maintain their reference must copy the object elsewhere. Subsequent users have the option of making their own copy, or updating their link to refer to one of the new copies.

A.1.4 Mobility

Objects may move from one site to another. If they do, a forwarding pointer must remain at the old location until the time-to-live expires. This enables anyone with a non-expired link to the object to refresh the link and record the object's new location.[5]

A.2 Directories

A directory is an object and as such, everything that applies to objects also applies to directories. The physical representation of a directory is interpreted by the Prospero server on the system storing the directory. The Prospero protocol defines the interface through which a directory is accessed.

[4]Despite this, well known names from agreed upon starting points might be entered as application-defined attributes for objects.

[5]To place a forwarding pointer, set the old object's FORWARDING-POINTER atribute. It must be manually moved to the new server at this time.

A.2.1 Directory Attributes

The following attributes are defined for directories:

DIRECTORY-ORDERING **(not yet implemented)** This attribute is a 3-tuple. The default value is

```
SORTED NAME-COMPONENT INCREASING
```

which means that, by default, directory entries are sorted in increasing order according to the value of their NAME-COMPONENT attribute.

The first element of the tuple is `SORTED` or `UNSORTED`. If `UNSORTED`, the next two elements must be null. If `SORTED`, the second field specifies an attribute which is the sort key. The third element is either `INCREASING` or `DECREASING`, meaning the order of sorting.[6]

If the INCLUDE-NATIVE attribute is set to any value other than `NONATIVE`, then the DIRECTORY-ORDERING cannot be `UNSORTED`.

INCLUDE-NATIVE **(not yet implemented)** Whether to include information from the native filesystem in the directory. If files are included, they will be included from the real directory on the server with the same <**hsoname**> as the Prospero directory. Its <**attribute-type**> is a single-element `SEQUENCE`. Values are:

`NONATIVE` Do not include files from native directory.

`INCLNATIVE` Include all files from native directory.

`INCLREAL` Smae as above, but (for the UNIX server) do not include "`.`" and "`..`".

`NATIVEONLY` All entries in directory are from native dir; no links have been added. For the UNIX server, "`.`" and "`..`" are NOT included.

`PSEUDO` Directory is not real. This is set for all directories returned from filters and by Archie and Gopher.

A.2.2 Link Attributes

Prospero directories contain links to the objects that are included in the directory. The following information is maintained for each link.

NAME-COMPONENT This is the single component of the object's path relative to the current directory. Its <**attribute-type**> is `SEQUENCE`.

Short description of link (Optional).

LINK-TYPE . The type of a link can be either normal [L] or union [U]. In the case of a normal link, an entry for the link is visible when the directory is listed. In the case of a union link, which can only be made to another directory, the links from the target directory appear as part of the directory from which the union link originates when the originating directory is listed. If

[6] In the current implmentation, the only sort keys that may be specified must have an <**attribute-type**> of `ASCII`.

multiple objects have the same name, the order in which the union links appear determines which object is visible.

Two types of links which may appear locally (either in the server, or on the client) are [-] deleted, and [N] native. It is not legal for these codes to be sent across the network. Type [N] links are converted to [L] and type [-] are skipped entirely.

TARGET Target of link: For links in a directory that don't point to objects, could be `SYMBOLIC` or `EXTERNAL`. For a forwarding pointer it is `FP`. Links to objects will have a target field indicating the cached value of the object's BASE-TYPE attribute. When stored on a vlink structure, it may have exactly one of the following forms: `OBJECT`, `FILE`, `DIRECTORY`, `DIRECTORY+FILE`. There is no guarantee that the type of the target has not changed. Thus, this field is the cached value of the object's base type. For union links, this field is always `DIRECTORY` or `DIRECTORY+FILE`. See section 3.5.3 for a further discussion of this.

Hidden/Not-Hidden/Externally-Hidden[7] By default, a hidden link is not displayed when a directory is listed. It is, however, returned by the directory server, and is traversed if the actual name is specified in a pathname. An Externally-Hidden link is a hidden link that will be displayed if the current virtual system is the same as the owning virtual system for the directory containing the link. The user may override the hidden option, causing hidden links to be listed. Note that it is also possible to hide a link by specifying its protection as non-listable. Such links will **only** be returned by the directory server when the actual name of the link has been specified.

FILTER Multiple filters allowed. The filter attribute is a pointer to a program that can be used to filter the contents of directories to which links are made. *data* is the set of optional arguments passed to the filter program. In addition to the linked directory and arguments, the filter has access to all other information available from the current directory.

Filters come in several types. By default, a filter is a directory filter, and is applied when searching directories. A hierarchy filter is similar to a directory filter. The difference is that a directory filter is applied to a single directory, while a hierarchy filter is applied to the entire hierarchy (including subdirectories) reached through a link. Directory and hierarchy filters come in two types. The default is post-expansion. The filter is applied after all union links have been expanded. A pre-expansion filter is applied before union links are expanded. An object filter is applied when accessing an object other than a directory, and might be used, for example, to cause some operation to be performed on the object before it is accessed. Note, however, that all types of filters are associated with links.

Filters are applied to a directory in the order of pre-or post expansion, decreasing depth of the links to which they are attached (for hierarchy filters), and finally in the order that the filters are specified on the traversed links.

Attributes (optional). Application attributes to be associated with the link. This allows a user or application to add (or override) attributes to the linked object (which might be owned by someone else, and thus not modifiable).

HOST The name of the host on which the object can be found.

HOST-TYPE The type of the hostname. In particular, whether the hostname is an Internet address, a domain style name, or a name in some other naming system.

Note that a symbolic link is a link where the destination host is a virtual system, and the destination object name is a name relative to that virtual system.

HSONAME Host-specific object name. The name of the object relative to the destination system.

HSONAME-TYPE The type of HSONAME. Different types of names include numerical file IDs, names relative to the root of the local file system, etc. Right now, only pathnames are implemented, and their type is `ASCII`.

OBJECT-VERSION By specifying a version number in a directory link, the link is made to a specific version of the object, and changes to the object will not be visible through the link.

Access control info (optional). Allows restrictions on who can read or modify individual directory entries. These are really attributes, though they are in fact retrieved and modified with LIST-ACL and EDIT-ACL, and do not appear on the normal attribute listings.

DEST-EXP **Destination expiration date and time**. Its implied type is a single-element `SEQUENCE`, in ASN-TIME format.[8] This entry indicates how long the information in the link should be considered valid. When an object is accessed through a link, the destination expiration date should be set to the current time plus the destination time-to-live.

Note that expired directory entries do not disappear. Typically they remain valid. The expiration means that there is no longer a guarantee that the object originally referenced can still be found.

ID. When a new object is created, a unique object identifier may be assigned. This identifier can be included in links, and used to further verify that the named object is the one that is actually desired. It allows one to reference objects after their expiration dates with the guarantee that if these identifiers match, it is the same object. In the prototype, the object identifier is a random number of type REMOTE. Other types of object identifiers will be defined after more work is done by an IETF working group.

Valid-till (optional). If a link is a cached value, then this field indicates how long the entry should be considered valid. For example, a symbolic link may have a corresponding cached hard link. Until its expiration a cache entry may be used instead of searching based on the symbolic link. If this field is 0, then the link is not a cached entry.

Last-update (optional). This is the time the link was last updated. Its expected use is for resolving conflicting updates in replicated directories.

[8] Implementers should use the `asntotime()` function in the `pfs` library in order to convert an ASN-TIME string into a UNIX timestamp, and the `timetoasn()` function to perform the reverse conversion. Applications writers are encouraged to use ASN-TIME format to represent all timestamps sent across the network.

A.2.3 Replication

When support for replication is added to Prospero, a directory might include multiple links with the same name for the same object. Not all replicas need be listed, however, because each replica will maintain its own list of siblings. Multiple entries are important to increase reliability and availability.

Appendix B

Asynchronous Reliable Delivery Protocol

This appendix describes the asynchronous reliable delivery protocol (ARDP) used by Prospero. As used by Prospero, this protocol is layered on top of the Internet User Datagram Protocol.

Prospero implements its own ARDP because at the time of initial development we were unable to find any that were suitable for general use. Most systems that use any sort of reliably delivered message protocol implement their own around the specific needs of their application. Like these other systems, early versions of the Prospero protocol defined the mechanisms needed for retries and packet sequencing. As these mechanisms were refined, the functionality was moved to a separate protocol layer (and is implemented as a separate library) to improve modularity, and in hope that this general and simple ARDP can be used for other purposes.

The ARDP protocol is designed for a request/response style of interaction, where a client sends a request message to a server in one fell swoop and receives a reply message from the server. The server can send the packets composing the reply message slowly, as data becomes available, while it is still processing the rest of a reply. In the current implementation, each request message sent by a machine from a particular port has its own connection id.

ARDP was designed so that in the common case, the additional overhead of guaranteeing reliability is as small as possible. Unless special processing is required, the header is kept small, and unless a packet is lost, no additional packets are sent. If a field is not specified, the default value is used in its place. All fields up to and including the last field specified must be filled in, but the header may be truncated at any point, after which all remaining fields take on their default values. The ARDP header contains the following fields:

Octet 0 Version and header length: High order two bits are ARDP version number mod 4 (this is version 0). Low order six bits are the header length including octet 0.[1]

Octets 1–2 Connection ID: Defaults to zero. It must be specified in the response to any request that specified a non-zero connection ID.

Octets 3–4 Packet number: Defaults to 1 if not specified. A specified value of 0 indicates an unsequenced control packet which should not be passed to the application. Note that unsequenced

[1] The length of the total packet, including data, is available via the UDP layer, as are the port and IP address of the sending host.

control packets cannot request acknowledgements, nor is there any way for the sender of such a packet to be sure that they have arrived.

Octets 5–6 Total number of packets in this message: Defaults to 0 if not known, or retains current value if it was provided in any earlier messages. If the packet number was also not specified, then it defaults to 1. A specified value of 0 means use the default.

Octets 7–8 Received through: Sequence number through which all packets have been received by the sender of this packet. Defaults to current value if specified in previous message. Defaults to 0 otherwise. The recipient's count of packets received through is normally monotonically increasing; this keeps the count from being set backwards in case an out-of-order packet is received. However, if the "reset received through" option (option 2) is specified in octet 12, then it means reset to 0 (i.e. it forgot or lost the earlier messages). More generally, specifying any explicit value for this field along with the "reset received through" option resets the peer's count, possibly backwards. The recipient should not set its internal value of this field backwards unless the "reset received through" option is set.

Octets 9–10 Wait (expected time till response): Defaults to current value. Specified value of 0 means revert to client-specified backoff algorithm. Specifying a non-zero value lets the client know that a request might not be processed for some time and that the client should not retry the request until the specified time. The client may retry sooner if it believes messages are available which have been missed (e.g., gaps in the list of received packets). This is an unsigned quantity, measured in seconds, in network octet order (i.e., octet 9 is more significant than octet 10). A specified value of 65,535 ($FFFF_{16}$; all bits turned on) means greater than or equal to 65,535 seconds until the next packet. (We do not expect that this value will ever be used, but it is defined for the sake of completeness.)

Octet 11 Flags: Octet 11 is a bit vector specifying option flags. The flags may themselves require additional fields specific to the flag. These fields appear at the end of the header in the order they are needed when reading flags from the low order bit to the high order bit, followed by any extra fields needed by the flag specified by the 12th octet.

Value of flags for octet 11

Bit No.	Meaning	Additional Fields
0 (low order)	Additional Address Information Follows	Variable length (see below)
1	Priority Follows	2 octets (see below)
2	A Protocol ID for a higher-level protocol follows	2 octets (see below)
3–5	Unused	
6	This packet is a sequenced control packet only; it should not be returned to the application by the ARDP library.	None
7 (high order)	Please Acknowledge this Packet	None

Octet 12 More Options: Octet 12 specifies exactly one (1) of up to 256 other options. The options may themselves require additional fields specific to the options. (See discussion at Octet 11).

Value of options for octet 12

Value	Meaning	Additional Fields
0	No Option Specified	None
1	Client to server: Cancel Request. Server to client: Connection refused.	None
2	Reset peer's received-through count.	Specified in octets 7–8
3	Packets received beyond received-through	The rest of the header after additional data for octet 11 flags is an arbitrary number of octets. These are bit-vectors specifying which packets beyond the received-through specified in this packet have been received by the sender of this packet. For example, if the received-through is set to 43, then we know that packet 44 has not been received. The low order bit of the first octet of the additional field will be turned on if packet 45 has been received, and off if it has not. The high-order bit will be turned on if packet 52 has been received, and off if it has not. Similarly, the low-order bit of the second octet of additional information will be turned on if packet 53 h as been received, and so on. The recipient of this information may choose to ignore it and use a simpler resend strategy. Similarly, this information is never required to be sent.
4	Redirect (used by servers): The client should send any unacknowledged packets already sent and all subsequent packets in this message to a new addresss. This is designed to be used as a load-shedding device. In one common case, this will be the entire response a server gives to a request, and the client will resend the entire request to a new server; in the other common case, this will be used in conjunction with option 6 or 7.	6 octets. The first 4 octets are the IP address of the new server, in network byte order. The next 2 octets are the UDP port to which the request should be sent, also in network byte order.
5	Redirect and notify (used by servers). Like option 4, but the client's network layer should also notify its caller that all subsequent requests intended for the old server should be sent to the new server instead.	Same as option 4.
6	Forwarded: This request was received from a client, and the sender is a server forwarding it to the recipient for processing. The recipient should pretend that it received this message from the sender indicated by the additional fields, not from the real sender of this message. (If implemented, this request should be accepted only from one of a group of trusted hosts.) This option is intended to be used by a central server which distributes requests to several subsidiary servers which do the actual work of processing the request, but which use the central server as a contact point. Presumably, it is cheaper for the central server to forward the request to the subsidiary servers over a local area network rather than for the client (who may be quite far away) to retransmit it. The central server has done the job of notifying the original client (through option 4 or 5) that further requests and retransmissions should go to the new server.	6 octets: The IP address and port of the original sender of this message, as in number 4.

Value of options for octet 12

7	Forwarded; Please notify: Like option 6, but the receiving server should notify the client of the switch (through option 4 or 5).[2]	6 octets: The IP address and port of the original sender of this message, as in number 4.
8-252	Undefined	Undefined
253	Request Queue Status	1 octet. If bit 0 (low order) is set, the position in the queue is requested. If bit 1 is set, the estimated time until this request will be completed is requested. The recipient may ignore this option.
254	Response to 253.	1 octet of flags, followed by 1 or 2 additional fields. If bit 0 (low order) of the flags is set, the position in the queue is returned as a 2 octet network byte order representation of an unsigned quantity. A value of $FFFF_{16}$ (all bits turned on) means a queue position of $FFFF_{16}$ or further. (We do not expect this value to ever be used, but it is included for the sake of completeness.) If bit 1 of the flags is set, the estimated time until this request will be completed is returned as a 4-octet network byte order unsigned value, representing a time in seconds. A value of $FFFFFFFF_{16}$ (all bits turned on) means a time of $FFFFFFFF_{16}$ seconds or more. (We do not expect this to ever be used).
255	Reserved for future expansion	Undefined

Octets 13 and above Fields specific to particular flags and options.

First, additional data fields specific to the flags in octet 11 should be specified.

Next Octets Additional Address Information (if Additional Address Information flag specified): The first octet specifies the type of additional address information. The next octet specifies the length of the address information, from 0 to 255 octets.[3] Its length does not include the two octets that specify type and length. The following octets contain the address information itself, and its format is dependent upon the type of address information.

Next 2 octets Priority (if Priority flag specified): These octets are a signed integer representing the priority of the request. Not all implementations understand this message, and many

[2] You might think that the recipient needs to worry about the IP address of the FORWARD message it sends to the original client being different from that of the original server (the sender of this message). However, this is not the case. The existence of multi-homed hosts means that the sender of a message cannot assume that the IP address of a reply (as recorded in the UDP packet encapsulating the response it receives) is the same as the address the packet was sent to. The sender must use the connection ID to match up sent messages and received replies.

[3] The entire 255 octets are not available for address information, since they are part of the header, and the maximum header length header is limited to 64 octets.

that do will not honor requests for expedited handling. Negative numbers indicate expedited handling while higher numbers indicate greater delays. A priority of 0 is normal.

Next 2 octets Protocol ID (if Protocol ID flag specified): These octets identify the interpretation of the data carried in the packet. The default, or an explicitly specified value of 0, mean that it is not specified, but has been agreed upon externally (i.e. the applications will know).

Next octets Any data specific to the option set by octet 12 should be specified. This is the data specified in the "additional fields" column of the table "Value of flags for octet 12."

Appendix C

Prospero Conventions

C.1 <hsoname>s

There are some special <**hsoname**>s that the current Prospero server may recognize, if it has been configured appropriately. If an <**hsoname**> begins with the string `AFTP/`, it is interpreted as a path relative to the anonymous FTP directory on that server. If an <**hsoname**> begins with the string `GOPHER`[`(`<**gopher-port-number**>`)`], it is interpreted to refer to GOPHER data files on that host . If an <**hsoname**> begins with the string `GOPHER-GW`, it refers to GOPHER data files on another host. If an <**hsoname**> begins with the string `ARCHIE/`, it represents an Archie database query.

Appendix D

Prospero Protocol Changes from Version 1 to Version 5

Versions 2, 3, and 4 of the Prospero protocol were not used. The protocol number jumped from version 1 to version 5 to keep the software number and version number consistent; the first version of the server to implement version 5 protocol had a software version number of 5.

For now, all Prospero servers continue to accept Version 1 of the protocol.

Not all of the protocol changes are listed here.

In version 1, the `EDIT-LINK-INFO` command was called `MODIFY-LINK`. The syntax of the arguments has also been changed.

The method of specifying attributes to `LIST` has changed. The lines returned from `LIST` are also in a new format.

The quoting mechanism has been formalized, and extended to apply to multiple-component directory names. Therefore, the `ONECOMP` option to `LIST` is now officially dead (it was never implemented in any server anyway). Note that no Version 1 server ever implemented quoting properly anyway. Therefore, if a client speaks Version 1 Protocol, it will not be able to access filenames with spaces in them.

The **<magic-no>** field has been replaced with zero or more occurrences of the sequence `ID` **<id-type>** **<id-value>**. `GET-OBJECT-INFO` used to have the reserved keyword `ID`, but this has been flushed.

The syntax of the arguments of `EDIT-OBJECT-INFO` has been changed, in order to avoid a syntactic amgiguity.

Many changes have occurred to the arguments of commands.

Filter syntax has changed drastically.

Separate principals are now sent as separate protocol tokens in ACL and attribute definitions.

The TARGET field used to have a possible value of `SYM-LINK`. This has been changed to `SYMBOLIC`. The BASE-TYPE field has been introduced.

The syntax of a **<target>** of `EXTERNAL` has changed radically. EXTERNAL links now look a lot more like conventional links. The meaning of the ACCESS-METHOD attribute has changed radically; it's now a lot more flexible.

Support for Kerberos has been specified.