Yellow Pages
Protocol Specification

Yellow Pages
Protocol Specification

1. Introduction and Terminology

The Yellow Pages (YP), Sun's distributed lookup service, is a network service providing read access to a replicated database. The lookup service is provided by a set of YP database servers, which communicate among themselves to keep their databases consistent. The client interface to this service uses the Remote Procedure Call (RPC) mechanism.

Translating or mapping a name to its value is one of the most common operations performed in computer systems. Common examples are the translation of a variable name to a virtual memory address, the translation of a user name to a system ID or list of capabilities, and the translation of a network node name to an internet address. There are two fundamental read-only operations that can be performed on a map: matching and enumeration. Match means to look up a name (which we call a **key**) and return its current value. Enumerate means to return each key-value pair in turn.

The YP supplies matching and enumeration operations in a network environment, in which high availability and reliability are required. It provides that availability and reliability by replicating both databases and database servers on multiple nodes within a single local net, and within the internet. The database is replicated, but not distributed: all changes are made at a single server and eventually propagate to the remaining servers without locking. The YP is appropriate for an environment in which changes to the mapping databases occur on the order of tens per day.

The YP operates on an arbitrary number of map databases. Map names provide the lower of two levels of a naming hierarchy. Maps are themselves grouped into named sets, called **domains**. Domain names provides a second, higher level of naming. Map names must be unique within a domain, but may be duplicated in different domains. The YP client interface requires that both a map name and a domain name be supplied to perform match and enumeration operations.

The YP achieves high availability by replication. One area not addressed by the protocol which has to be addressed by the implementors is global consistency among the replicated copies of the database. Every implementation should be designed so that at steady state a request yields the same result when it is made of any YP database server. Update and update-propagation mechanisms must be implemented to supply the required degree of consistency.

1.1. RPC — Remote Procedure Call

Sun's Remote Procedure Call (RPC) mechanism defines a paradigm for interprocess communication modeled on function calls. Clients call functions that optionally return values. All inputs and outputs to the functions are in the client's address space. The function is executed by a server program.

Using RPC, clients address servers by a program number (this identifies the application level protocol that the server speaks), and a version number. Additionally, each server procedure has a procedure number assigned to it.

In an internet environment, a client must also know the server's host internet address, and the server's rendezvous port. The server listens for service requests at ports that are associated with a particular transport protocol — TCP/IP and/or UDP/IP.

The format of the data structures used as inputs to and outputs from the remotely-executed procedures are typically defined by header files that are included when the client interface functions are compiled. Levels above the client interface package need not know any particulars of the RPC interface to the server.

1.2. XDR — External Data Representation

The Sun External Data Representation (XDR) specification establishes standard representations for basic data types (such as strings, signed and unsigned integers, and structures and unions) in a way that allows them to be transferred among machines with varying architectures. XDR provides primitives to encode (that is, translate from the local host's representation to the standard representation) and decode (translate from the standard representation to the local host's representation) basic data types. Constructor primitives allow arbitrarily complex data types to be made from the basic types.

The YP's RPC input and output data structures are described using XDR's data description language. In general, the data description language looks like the C language, with a few extra constructs. One such extra construct is the *discriminated union*. This is like a C language union, in that it can hold various objects, but differs from it in that a discriminant indicates which object it currently holds. The discriminant is the first thing across the wire. Consider a simple example:

```
union switch (long int) {
    1:
        string exmpl_name<16>
    0:
        unsigned int exmpl_error_code
    default:
        struct {}
}
```

The example should be interpreted as follows: the first object to be encoded/decoded (that is, the discriminant) is a long integer. If it has the value one, the next object is a string. If the discriminant has the value zero, the next object is an unsigned integer. If the discriminant takes any other value, don't encode or decode any more data.

A *string* data type in the XDR data definition language adds the ability to specify the maximum number of elements in an byte array or string of potentially variable size. For instance:

```
string domain<YPMAXDOMAIN>;
```

states that the byte sequence *domain* may be less than or equal to `YPMAXDOMAIN` bytes long.

An additional primitive data type is a *boolean*, which takes the value one to mean `TRUE` and zero to mean `FALSE`.

2. YP Data Base Servers

2.1. Maps and Operations on Maps

2.1.1. Map Structure

Maps are named sets of key-value pairs. The keys and their values are counted binary objects. The keys and their values may be ASCII information, but they need not be. The data comprising a map is determined by the client applications that are the final customers for the data, not by the YP. The YP has no syntactic nor semantic knowledge of the map contents. Neither does the YP determine or know any map's name. Map names are managed by the YP's clients. Conflict in the map namespace must be resolved by human administrators outside the YP system.

Typical implementations for YP maps are files or DBMS systems. The design of the YP's map database is an implementation detail, and is unspecified by the protocol.

2.1.2. YP Private Key Symbols

It is useful to be able to embed key-value pairs that may be used by the YP subsystem itself, or by human administrators or administration programs within all maps. Keys beginning with **YP_** may be conventionally used to embed out-of-band information within a map, and should be considered to be YP-private. The client interface to the YP's enumeration functions should be implemented to filter out YP-private keys. Client programs should not see them; they won't know what to do with them, and client parsers should not be forced to do the filtration.

A unfiltered interface to the YP enumeration functions may also be supplied for programs that need to see YP-private keys. Alternatively, it could be assumed that any client that needs to see a YP-private key knows the name of that key. If that assumption is made, the YP match operation is sufficient, and no unfiltered flavor of the YP enumeration operations needs to be supplied.

The price paid for the ability to imbed administrative information within maps is that the key namespace is reduced.

2.1.3. Match Operation

The YP supports an exact match operation in the `YPPROC_MATCH` procedure. That is, if a match string and some key in the map are exactly the same, the value of the key is returned. No pattern matching, case conversion, or wildcarding is supported.

2.1.4. Map Entry Enumeration Operations

The two operations which exist to enumerate the entries of a map are a ''get first key-value pair'' operation (the `YPPROC_FIRST` procedure), and a ''get next key-value pair'' operation (the `YPPROC_NEXT` procedure). If ''get first'' is called once, and then ''get next'' is called until the return value indicates that there are no more entries in the map, each entry in the map will be seen exactly once. Further, if the same sequence of calls is made again on the same map at the same YP database server, the order in which the entries will be seen is the same.

The actual ordering function is unspecified, and may not be assumed. It also may not be assumed that enumerating a map at a different YP database server will return the entries in the same order, whether that server represents the same implementation or not.

2.1.5. Map Update

The update of YP maps is an implementation detail which is outside the specification of the YP service.

2.2. Master and Slave YP Data Base Servers

The protocols assume that for each map there is one distinguished YP database server, called the map's *master*. Map updates take place only on the master. An updated map should be transferred from the master to the rest of the YP database servers, which are *slave* servers for this map.

It is possible for each map to have a different YP database server as its master, or for all maps to have the same master, or any other combination. The choice of how to set up map masters is one of implementation and administrative policy.

2.3. Map Propagation, and Consistency

Getting map updates from the master to the slaves is called map propagation. Neither technology nor algorithms for map propagation are specified by the protocol. Map propagation may be entirely manual: for instance, a person could copy the maps from the master to the slaves at a regular interval, or when a change is made on the master. This is unnecessarily labor intensive. There are hooks within the protocol for automatic convergence. The procedures designed for server-to-server communications are described in the next section.

In order to escape from the idiosyncrasies of any particular implementation, all maps should be uniformly timestamped internally. An internal timestamp allows the map to be copied to or reconstructed at any number of nodes, without the time format, local clock time, or file creation or modification algorithms at that site having any effect on the map's version.

The timestamp should be created at the site where the map was created, or was last modified. The timestamp is out-of-band data, as far as the applications using the map are concerned, and should be associated with the YP-private key `YP_LAST_MODIFIED`. Its value should be an ASCII numeric sequence representing the time the map was created or last modified as the number of seconds since January 1, 1970 (GMT). The ASCII numeric sequence may be zero-padded to the left, up to a total length of ten characters. Each YP database server can read the `YP_LAST_MODIFIED` entry from each map it serves, and compare it with the version its peers have.

The intent is for a slave to try to get the current copy from the master. If the master is unreachable, the subnet can still converge at the highest available order number. The slaves communicate among themselves to guarantee that all agree on the current version.

2.3.1. Functions to Aid in Map Propagation

Any YP database server can communicate with any other. Any server may call `YPPROC_MATCH`, `YPPROC_FIRST`, or `YPPROC_NEXT` in a second server, in which case the first server is a client of the second. The protocol also has four functions that exist to help servers converge on a single version of a map.

`YPPROC_GET` is called by a master server in a peer slave server. It tells the slave server to get a new version of a map from the master.

`YPPROC_PUSH` is called by an administrative program in a master. It tells the master to notice that a new version of the map exists, and tell the peer slaves to get the new version.

YPPROC_PULL is called by an administrative program in a slave. It tells the slave to get a new version of a map.

YPPROC_POLL can be called either by a server or by an administrative program in any server. It is called to find out what the server's current map version is, and which server it thinks is the map's master.

2.3.2. Map Transfer Mechanism

The way a map is transferred from one server to another is not specified by the protocol. One possibility is the manual process described above. Another might be that a YP database server could activate some other process that would exist only to do the map transfer. A third might be for a server to enumerate the more recent version of the map, by using the normal client map enumeration functions.

If the enumeration method is used, it will take several functions to transfer the whole map, and the map version may change at the supplying site. A version change over the lifetime of the transfer can be detected by the consumer server if the consumer brackets the enumeration with calls to the YPPROC_POLL procedure in the supplier.

2.4. Domains

Domain provide a second level for naming within the YP subsystem. They are names for sets of maps, therefore create separate map name spaces. Domains provide an opportunity to break large organizations up into administerable chunks, and the ability to create parallel, non-interfering test and production environments.

Ideally, the domain of interest to a client ought to be associated with the invoking user, but in practice it is useful for client machines to be in a default domain. Implementations of the YP client interface should supply some mechanism for telling processes the domain name they should use. This is needed not only because the concept of domain is a useless one as far as most programs are concerned, but, more importantly, so that programs can be written that are insensitive to both location and the invoking user.

Information logically associated with all domains (or to no domain) can be held in a domain that is really a meta-domain. This domain may have a well-known name, so that information within it can be accessed regardless of the machine's default domain, or of the domain of the invoking user.

2.5. Non-features

The following capabilities are not included in the current YP protocols:

2.5.1. Map Update Within the YP

All write (and delete) access to the YP's map database is assumed to be outside of the YP subsystem. It is probable that write access to the map database will be included in later versions of the YP protocols.

2.5.2. Version Commitment Across Multiple Requests

The YP protocol was designed to keep the YP database server stateless with regard to its clients. Therefore, there is no facility for contracting with a server to preallocate any resource beyond that required to service any single request. In particular, there is no way to get a server to commit to use a single version of a map while trying to enumerate that map's entries.

2.5.3. Guaranteed Global Consistency

There is no facility for locking maps during the update or propagation phases, therefore it is virtually guaranteed that the map database be globally inconsistent during those phases. The set of client applications for which the YP is an appropriate lookup service is one that (by definition) must be tolerant of transient inconsistencies.

2.5.4. Access Control

The YP database servers make no attempt to restrict access to the map data by any means. All syntactically correct requests are serviced.

2.6. YP Data Base Server Protocol Definition

This section describes version 1 of the protocol. It is likely that changes will be made to successive versions as the service matures.

2.6.1. RPC Constants

All numbers are in decimal.

YPPROG 100004
: The YP database server protocol program number.

YPVERS 1
: The current YP protocol version.

2.6.2. Other Manifest Constants

All numbers are in decimal.

YPMAXRECORD 1024
: The total maximum size of key and value for any pair. The absolute sizes of the key and value may divide this maximum arbitrarily.

YPMAXDOMAIN 64
: The maximum number of characters in a domain name.

YPMAXMAP 64
: The maximum number of characters in a map name.

YPMAXPEER 256
: The maximum number of characters in a YP server host name.

2.6.3. Remote Procedure Return Values

This section presents the return status values returned by several of the YP remote procedures. All numbers are in decimal.

```
typedef enum {
        YP_TRUE = 1,            /* General purpose success code. */
        YP_NOMORE = 2,          /* No more entries in map.  */
        YP_FALSE = 0,           /* General purpose failure code.*/
        YP_NOMAP = -1,          /* No such map in domain.*/
        YP_NODOM = -2,          /* Domain not supported.*/
        YP_NOKEY = -3,          /* No such key in map.*/
        YP_BADOP = -4,          /* Invalid operation.*/
        YP_BADDB = -5,          /* Server database is bad.*/
        YP_YPERR = -6,          /* YP server error.*/
        YP_BADARGS = -7         /* Request arguments bad.*/
} ypstat;
```

2.6.4. Basic Data Structures

This section defines the data structures used as inputs to and outputs from the YP remote procedures.

domainname

```
typedef string domainname<YPMAXDOMAIN>
```

mapname

```
typedef string mapname<YPMAXMAP>
```

peername

```
typedef string peername<YPMAXPEER>
```

keydat

```
typedef string keydat<YPMAXRECORD>
```

valdat

```
typedef string valdat<YPMAXRECORD>
```

ypmap_parms

```
struct ypmap_parms {
        domainname
        mapname
        unsigned long int ordernum
        peername
}
```

This contains parameters giving information about map *mapname* within domain *domainname*. The *peername* parameter is the name of the map's master YP database server. If any of the three string pointers represent unknown (or unavailable) information, the parameters will be null strings. The *ordernum* parameter contains a binary value representing the value of the map's `YP_LAST_MODIFIED` key. If the `YP_LAST_MODIFIED` value is unavailable, *ordernum* contains the value 0.

yprequest

```
struct yprequest {
        union switch (enum ypreqtype) {
                YPREQ_KEY:
                        struct {
                                domainname
                                mapname
                                keydat
                        }
                YPREQ_NOKEY:
                        struct {
                                domainname
                                mapname
                        }
                YPREQ_MAP_PARMS:
                        struct ypmap_parms
                default:
                        {}
        }
}
```

ypresponse

```
struct ypresponse {
        union switch (enum ypresptype) {
                YPRESP_VAL:
                        struct {
                                ypstat
                                valdat
                        }
                YPRESP_KEY_VAL:
                        struct {
                                ypstat
                                valdat
                                keydat
                        }
                YPRESP_MAP_PARMS:
                        struct ypmap_parms
                default:
                        {}
        }
}
```

2.6.5. YP Data Base Server Remote Procedures

This section contains a specification for each function that can be called as a remote procedure. The input and output parameters are described using the XDR data definition language. Whenever the input parameter is a `struct yprequest`, the *mapname* and *domainname* parameters fully specify the map.

2.6.5.1. Do Nothing (Procedure 0, Version 1)

```
0. YPPROC_NULL ( ) returns ( )
```

This does no work. It is made available in all RPC services to allow server response testing and timing.

2.6.5.2. Do You Serve This Domain? (Procedure 1, Version 1)

```
1. YPPROC_DOMAIN (domain) returns (servesp)
        domainname domain;
        boolean servesp;
```

The server returns `TRUE` if it serves the passed `domain`, and `FALSE` otherwise. This function allows a potential client to ascertain whether or not a given server supports a named domain.

2.6.5.3. Answer Only If You Serve This Domain (Procedure 2, Version 1)

```
2. YPPROC_DOMAIN_NONACK (domain) returns (servesp)
        domainname domain;
        boolean servesp;
```

The server returns TRUE if it serves the passed `domain`; otherwise it does not return. The intent of the function is that it be called in a broadcast environment, in which it is useful to restrict the number of useless messages. If this function is called, the client interface implementation must be written so as to regain control in the negative case, for instance by means of a timeout on the response.

Sun's current implementation currently does `return` in the FALSE case by forcing an RPC decode error.

2.6.5.4. Return Value of a Key (Procedure 3, Version 1)

```
3. YPPROC_MATCH (req) returns (resp)
        struct yprequest  req;
        struct ypresponse resp;
```

The type of the `req` must be `YPREQ_KEY`. This returns the value associated with the key `keydat`. The type of the `resp` is `YPRESP_VAL`. If the `ypstat` parameter in the `resp` has the value `YP_TRUE`, the value data are returned in `valdat`.

2.6.5.5. Get First Key-Value Pair in Map (Procedure 4, Version 1)

```
4. YPPROC_FIRST (req) returns (resp)
        struct yprequest  req;
        struct ypresponse resp;
```

The type of the `req` must be `YPREQ_NOKEY`. The *resp* is of type `YPRESP_KEY_VAL`. If the value of the `ypstat` is `YP_TRUE`, this returns the first key-value pair from the map named in the `req` to the `keydat` and `valdat` parameters. An empty map is indicated by `ypstat` containing the value `YP_NOMORE`.

2.6.5.6. Get Next Key-Value Pair in Map (Procedure 5, Version 1)

```
5. YPPROC_NEXT (req) returns (resp)
        struct yprequest req;
        struct ypresponse resp;
```

The type of the `req` must be `YPREQ_KEY`. The `resp` is type `YPRESP_KEY_VAL`. If the value of the `ypstat` is `YP_TRUE`, this returns the key-value pair following the key-value named in the `req` parameter to the `keydat` and `valdat` parameters within `resp`. If the passed key is the last key in the map, the value of `ypstat` is `YP_NOMORE`.

2.6.5.7. Return Map Parameters (Procedure 6, Version 1)

```
6. YPPROC_POLL (req) returns (resp)
        struct yprequest  req;
        struct ypresponse resp;
```

The type of the `req` must be `YPREQ_NOKEY`. The `resp` is of type `YPREQ_MAP_PARMS`. The YP server returns the order number (binary timestamp value) and master server name for the map. If the domain is not supported, the `domainname` is a null string. If the map is unknown, the `mapname` is a null string. If unknown, the `ordernum` parameter has the value zero. If unknown, the `peername` is a null string.

2.6.5.8. Tell Peers About New Map (Procedure 7, Version 1)

```
7. YPPROC_PUSH (req) returns ( )
        struct yprequest req;
```

The type of the *req* must be YPREQ_NOKEY. The master server rechecks the named map to make sure that the map parameters are up-to-date. It then calls the YPPROC_GET procedure in each reachable peer. If the server is not the master of the named map, it takes no action.

2.6.5.9. Get Latest Version of Map (Procedure 8, Version 1)

```
8. YPPROC_PULL (req) returns ( )
        struct yprequest req;
```

The type of the req must be YPREQ_NOKEY. The slave server attempts to get a more recent version of the named map from a peer. The master, if reachable, is checked first. If the master's version is not greater than the slave's version, the slave does not try any further. If the master's version is greater than the slave's, the slave attempts to transfer the map. If the master is not reachable, the slave attempts to find a greater version held at some other peer. If the server is the master of the named map, it takes no action.

2.6.5.10. Get New Map Version From Here (Procedure 9, Version 1)

```
9. YPPROC_GET (req) returns ( )
        struct yprequest req;
```

The type of the req must be YPREQ_NOKEY. The server assumes that the caller is the master of the map, and tries to get a new version from that master server. In terms of version numbers and peer reachability, it follows the course of action described for YPPROC_PULL. If the server is the master of the named map after replacing the master peer's name with the caller's name, it takes no action. That is, if a master calls YPPROC_GET in itself, it takes no action.

3. YP Binders

3.1. Introduction

In order that any network service be usable, there must be some way for potential clients to find the servers. This section describes the YP binder, an optional element in the YP subsystem that supplies YP database server addressing information to potential YP clients.

In order to address a YP server in an ARPA internet environment, a client must know the server's internet address, and the port at which the server is listening for service requests. No contract is negotiated between a YP server and a potential client, therefore the addressing information is sufficient to bind the client to the server.

Of the many possible ways for a client to get the addressing information, one alternative is to supply an entity to cache the bindings, and to serve that binding database to potential YP clients. The theory is that if finding the service takes a lot of work, allocate a specialist to do it, rather than burden every client with a job that is irrelevant to its real function. A YP binder only makes sense if it is easier for a client to find the YP binder than to find a YP database server, and if the YP binder can itself find a YP database server.

We make the assumption that a YP binder is present at every network node, and because of this, addressing the YP binder is easier than addressing a YP database server. The scheme for finding a local resource is implementation-specific, but given that a resource is guaranteed to be local, there may be some efficient way of finding it. We further assume that the YP binder can find a YP database server in some way, but that that way is either complicated or time-consuming to do. If either of these assumptions is untrue, then probably your implementation is not a good bet for a YP binder.

If a YP binder is implemented, it can provide added value beyond the binding: it can verify that the binding is correct and that the YP database server is alive and well, for instance. The degree of sureness in a binding that the YP binder gives to a client is a parameter that can be tuned appropriately in the implementation.

3.2. YP Binder Protocol Definition

This section describes version 1 of the protocol. It is likely that changes will be made to successive versions as the service matures.

3.2.1. RPC Constants

All numbers are decimal.

YPBINDPROG 100007
: The YP binder protocol program number.

YPBINDVERS 1
: The current YP binder protocol version.

3.2.2. Other Manifest Constants

All numbers are decimal.

YPMAXDOMAIN 64

The maximum number of characters in a domain name. This is identical to the constant defined above within the YP database server protocol section.

ypbind_resptype

```
enum ypbind_resptype {
        YPBIND_SUCC_VAL = 1,
        YPBIND_FAIL_VAL = 2
}
```

This discriminates between success responses and failure responses to a `YPBINDPROC_DOMAIN` request.

ypbinderr

```
typedef enum {
        YPBIND_ERR_ERR 1        /* Internal error */
        YPBIND_ERR_NOSERV 2     /* No bound server for passed domain */
        YPBIND_ERR_RESC 3       /* System resource allocation failure */
} ypbinderr
```

The error case of most interest to a YP binder client is `YPBIND_ERR_NOSERV`; it means that the binding request cannot be satisfied because the YP binder doesn't know how to address any YP database server in the named domain.

3.2.3. Basic Data Structures

This section defines the data structures used as inputs to and outputs from the YP binder remote procedures.

domainname

```
typedef string domainname<YPMAXDOMAIN>
```

This is identical to the domainname string defined above within the YP database server protocol section.

ypbind_binding

```
struct ypbind_binding {
        unsigned long int ypbind_binding_addr
        unsigned short int ypbind_binding_port
}
```

This contains the information necessary to bind a client to a YP database server in the ARPA internet environment. *ypbind_binding_addr* holds the host IP address (4 bytes), and *ypbind_binding_port* holds the port address (2 bytes). Both IP address and port address must be in ARPA network byte order (most significant byte first, or big endian), regardless of the host machine's native architecture.

ypbind_resp

```
struct ypbind_resp {
        union switch (enum ypbind_resptype status) {
                YPBIND_SUCC_VAL:
                        struct ypbind_binding
                YPBIND_FAIL_VAL:
                        ypbinderr
                default:
                        {}
        }
}
```

This is the response to a `YPBINDPROC_DOMAIN` request.

ypbind_setdom

```
struct ypbind_setdom {
        domainname
        struct ypbind_binding
}
```

This is the input data structure for the `YPBINDPROC_SETDOM` procedure.

3.2.4. YP Binder Remote Procedures

Like the YP procedures earlier, these procedures are described using the XDR data definition language.

3.2.4.1. Do Nothing (Procedure 0, Version 1)

```
0. YPBINDPROC_NULL ( ) returns ( )
```

This does no work. It is made available in all RPC services to allow server response testing and timing.

3.2.4.2. Get Current Binding for a Domain (Procedure 1, Version 1)

```
1. YPBINDPROC_DOMAIN (domain) returns (resp)
        domainname domain;
        struct ypbind_resp resp;
```

This returns the binding information necessary to address a YP database server within the ARPA internet environment.

3.2.4.3. Set Domain Binding (Procedure 2, Version 1)

```
2. YPBINDPROC_SETDOM (setdom) returns ( )
        struct ypbind_setdom setdom;
```

This instructs a YP binder to use the passed information as its current binding information for the passed domain.