Internet Engineering Task Force
Service Location Working Group

Scott Kaplan
FTP Software Inc.
March 1993

Resource Location Protocol

1. Abstract 1
2. Status of this memo 1
3. Introduction 1
3.1 Requirements 1
3.2 Related Work 2
3.3 Overview 2
3.3.1 Major Components 2
3.3.1.1 User Agent (UA) 2
3.3.1.2 Resource Agent (RA) 2
3.3.1.3 Binding Broker (BB) 2
3.3.2 Protocol Operation 3
3.3.2.1 Locating Resources without a BB 3
3.3.2.2 Locating Resources with a BB 3
3.3.2.3 BB<->RA Registration 4
4 Structure of Attribute Information 5
4.1 Resource Location Syntax 5
4.2 Attribute Functions 6
4.3 Database Structure 6
4.4 Authentication 7
4.5 Access Control 7
4.6 Standard Attribute 8
4.6.1 Description 8
4.6.2 Distinguished Attributes 8
4.6.2.1 Standard resource_type Values 8
5 Protocol Data Units 9
5.1 Packet Header 9
5.1.1 Header Flags 9
5.1.2 Resource Database Cookie 9
5.2 PDU Data Portion 9
5.2.1 Locate 9
5.2.1.1 Description 9
5.2.1.2 Addressing 10
5.2.1.3 Silly Broadcast Algorithm 10
5.2.1.4 UA Caching 11
5.2.1.5 Distinguished Attributes 11
5.2.1.6 Scaling Issues 11
5.2.1.7 RA Registration 11
5.2.1.8 Request (Multicast) 12
5.2.1.9 Request (Broadcast) 12
5.2.1.10 Response 12
5.2.2 Resource Query 12
5.2.2.1 Description 12
5.2.2.2 Request 13
5.2.2.3 Response 13
5.2.3 Dictionary Query 13
5.2.3.1 Description 13

5.2.3.2 Request 13
5.2.3.3 Response 14
5.2.4 Resource Database 14
5.2.4.1 Description 14
5.2.4.2 Request 14
5.2.4.3 Response 15
5.2.5 DB Admin (Database Add/Database Delete) 15
5.2.5.1 Description 15
5.2.5.2 Request 15
5.2.5.3 Response 15
6. Technical Issues 15
6.1 Resource Location without a User Interface 15
6.2 Binding Broker to Binding Broker Protocol 16
6.3 Connections 16
6.3.1 Choosing to implement connection-oriented 16
6.3.2 Negotiating the connection 16
6.4 Resource Location over IP 16
6.5 Magic Numbers 17
6.5.1 IANA Registered Numbers 17
6.5.2 Error Values 17
6.5.3 Enumerations used in PDUs 17
7 Glossary 17
8 Acknowledgments 18
9 Author's Address 18
10 References 18
11 To Do List 19

1. Abstract

This document specifies the Resource Location (ResLoc) protocol. The ResLoc protocol allows a resource to advertise itself on the network. It defines a set of network queries that determines the location and type of resources available. It defines a mechanism to provide the minimal configuration information to effectively use that resource. It provides the ability to gather the advertised information in a central depository (binding broker) to reduce network bandwidth and allow the protocol to scale to large networks.

2. Status of this memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.
It is available in both ASCII and PostScript formats. The figures are described in PostScript and are not included with the ASCII version of the document. Copies of the figures are available from the author. Please respond with comments to the srv-location@apple.com mailing list.
Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

3. Introduction

The complexity of network administration and configuration is proportional to the number of clients and resources on the network. To reduce this complexity, ResLoc eliminates the need to supply configuration information at the client.
The ResLoc protocol provides a means for clients to locate and configure themselves to use networked resources. The initial requests that the client software makes do not require any knowledge of the network or the resources on the network. The responses from the resource agents contain information that the user agent can use to form subsequent requests. In this way, the client can build a map of all the networked resources or locate a particular resource.

3.1 Requirements

1. No client-side configuration
2. Must scale to large administrative domains (e.g. thousands of clients and hundreds of resources)
3. Must *scale down* to small sites of one or just a few networks. That is, resources and clients in small sites should be trivial to install and configure, albeit at a cost of increased network bandwidth that would be unacceptable in a larger site.
4. Must not require any knowledge on the part of the user about the network topology
5. What configuration there is should be associated with the resource and it should reside in only one place in the network.
6. Must support well-known (i.e. globally defined, or standardized) description of resources as well as site-specific descriptions.
7. The network must minimally support either multicast throughout an internet or broadcast onto a single physical network. If only broadcast is supported, there must exist relay agents which forward bootp broadcast from one physical network to a server on a different physical network.

3.2 Related Work

There are other projects related to locating resources on a network. These include:

- The Resource Administration Platform at Legato
- Network Information System+ at Sun
- X.500 as defined by OSI
- DNS as defined by RFC1035
- Banyan's StreetTalk
- HP/Apollo's Cell Directory Structure
- Apple's Name Bind Protocol
- Xerox's Clearinghouse. After all, everything good was done first, then forgotten, at Xerox

3.3 Overview

3.3.1 Major Components

To understand the resource location protocol, it is first necessary to understand the elements running on hosts in the network that cooperate, using the protocol, to share resource information.

3.3.1.1 User Agent (UA)

The User Agent (UA) is the software on the client side which is looking for a resource. It is most likely implemented as a set of functions in a library. These functions can be called from a user interface program or directly from a program looking for a resource.

The UA is responsible for forming the protocol requests that query the network elements that have resource information. The UA's most salient feature is that it operates without configuration information. It learns everything it needs by forming protocol requests. These protocol requests require no information other than what the UA learned from responses to previous protocol requests.

3.3.1.2 Resource Agent (RA)

The resource agent (RA) has the authoritative copy of the resource information. A single RA can have resource information for many resources. The RA is likely to be a process running on a host in the network.

The RA shares resource information with UAs by responding to protocol requests. The responses range from simply identifying the RA's network address to detailed information regarding a particular resource, depending on the protocol request that is received.

There is a specification of the abstract structure of the RA's database. The protocol is specified with respect to this abstract definition. However, the actual implementation of the RA database is not defined. The resource database is the only place in the ResLoc system that authoritative configuration information exists.

3.3.1.3 Binding Broker (BB)

The binding broker (BB) is a place for RAs to register the information in their resource database. It is a coalesce point for all the RA databases in the network. It provides a means for the UA to get responses to ResLoc requests without querying multiple RAs.

The BB is likely to be implemented as a process running on a host in the network. The implementation is probably similar to the RA since it is responding to protocol requests as a proxy for multiple RAs.

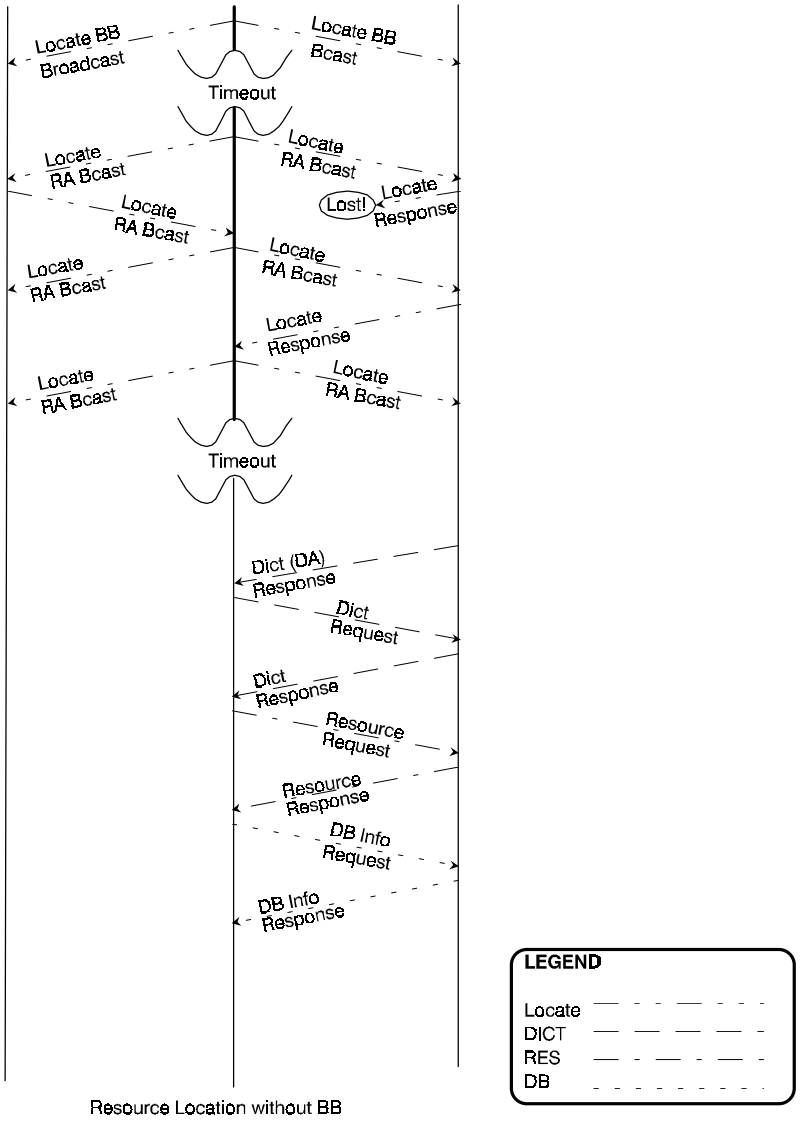
The BB can be thought of as a cache, however, since RAs register with the BB, the UA can trust responses from the BB as though they had come directly from the RA..

The BB gets all its configuration information from the RA using ResLoc protocol requests. It does not require any separate configuration.

<put network architecture picture here>

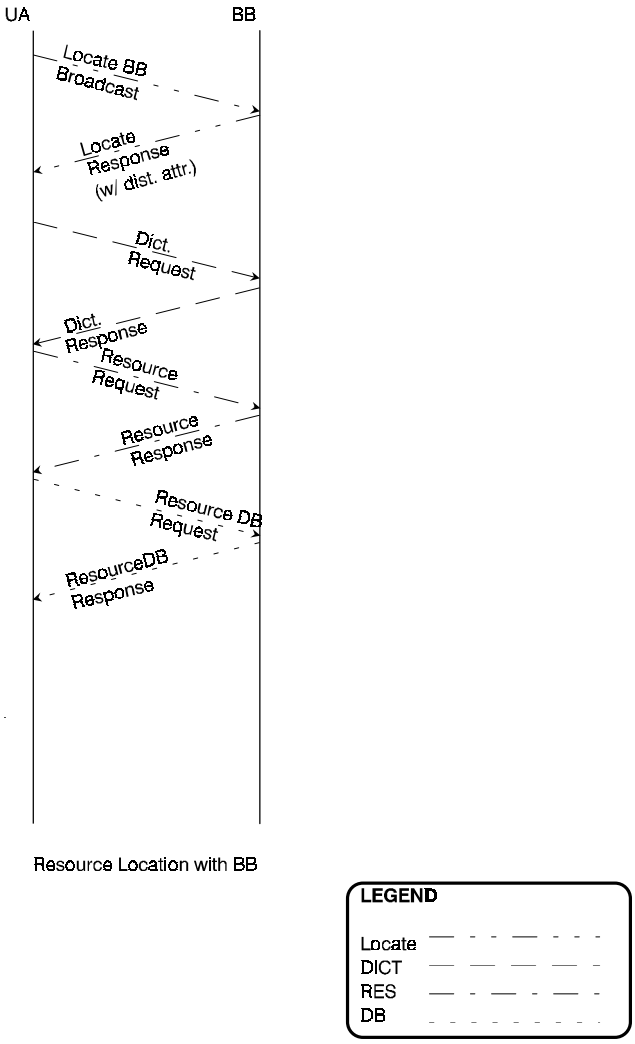
3.3.2 Protocol Operation

3.3.2.1 Locating Resources without a BB



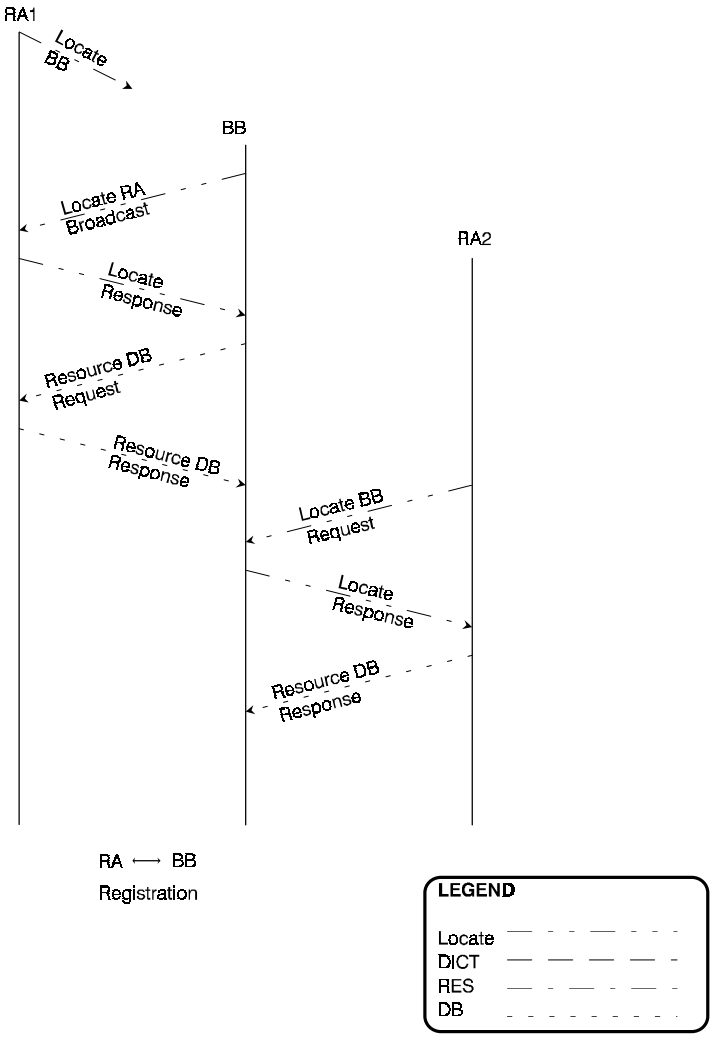
Resource Location without BB

3.3.2.2 Locating Resources with a BB



Resource Location with BB

3.3.2.3 BB<->RA Registration



4 Structure of Attribute Information

4.1 Resource Location Syntax

<attr expr> ::=	<attr template> |
	<bool op> <attr expr> <attr expr> |
	<not> <attr expr> |
<bool op> ::=	<and> | <or>
<and> ::=	'&'
<or> ::=	'|'
<not> ::=	'!'
<attr template> ::=	<attr class> <template value> <std attr>
<template value> ::=	<cond op> <integer value> |
	'=' <string template> |
	'=' <attr function>

```
<string template> ::=    <length> [ <wild card> ] <characters> [ <wild card> ]
<cond op> ::=            '>' | '<' | '=' | '=<' | '>='
<wild card> ::=          '*'

<attribute> ::=          <attr class> <attr op/value> <std attr> |
<attr op/value> ::=      <cond op> <integer value>
                         '=' <string value>
                         '=' <attr function>
<attr class> ::=         <string value>
<string value> ::=       'S' <string>
<integer value> ::=      'I' <integer>
<attr function> ::=      'F' <string>
<std attr> ::=           <boolean>
<boolean> ::=            <true> | <false>

<acl> ::=                <num users> <user list>
<num users> ::=          < 2 octets>
<user list> ::=          <user list> ',' <user> | NULL
<user> ::=               <string>

<true> ::=               '1'
<false> ::=              '0'

<string> ::=             <length> <characters>
<characters> ::=         ?
<length> ::=             <octet>
<integer> ::=            <4 octet>

<address> ::=            <addr type> <length> <value>
<addr type> ::=          <2 octets>
<value> ::=              <variable number of octets>
```

- Spaces, tabs, carriage returns, and line feeds are optional (i.e. ignored by the recipient) and, since they take up bandwidth, are discouraged.
- All string comparisons are caseless.
- <wildcard> matches 0 or more characters.

4.2 Attribute Functions

An attribute can have a function as a value. The RA is responsible for evaluating the function designated by the class name and using the returned value in evaluating the attribute expression. The semantics of the function are local to the RA.
The *<attr function>* is similar to *<string value>* except that the *<string>* is generated by the RA, rather than configured by an administrator. The requester which uses an *<attr function>* rather than a *<string value>*, should know that the value is subject to change rather quickly. An example of an attribute function is *online=yes*. In this case, the RA has a piece of code which determines whether or not the printer is on-line and sets the *<attr function>* accordingly.

4.3 Database Structure

The ResLoc protocol can be thought of as a method for retrieving data from a network distributed database. The RAs answer requests by supplying information from their individual databases. The protocol defines the transactions that allow a client to build a coherent picture of the distributed data.
To specify the protocol requests which retrieve information from the RA, we specify what the abstract structure of the database is that the RA is accessing. Note that this does not restrict or specify how the database is to be implemented. It describes the data and its structure abstractly so that the protocol can make assumptions about what information is available and how it can be retrieved.
The RA's attribute database can be thought of as a table of attributes with additional fields of data for each attribute. The fields of the table are:

Field Description	Field Data type
class	string
data type	enum: {STRING | INT}
enumerated type?	boolean
value	string or integer
standard attribute?	boolean
help string	string
configuration string	string
access control list	array of users

For example, the attribute database for the resource agent for a modem might look like this:

Database ACL: ringo(RW), paul(R)

Class	**Type**	**enum?**	**Value**	**std attr?**	**help string**	**config string**	**ACL**
"res_type"	STRING	N	"modem"	Y	"uses phone"	"128.127.50.1"	john, george
"baud"	INT	Y	9600	Y	NULL	"ATS50=6"	george
"baud"	INT	Y	2400	Y	NULL	"ATS50=3"	
"compress"	STRING	Y	"V.42"	N	NULL	"ATS180=2"	john
"MNP"	INT	N	4	N	NULL	"ATS180=3"	john, george

4.4 Authentication

In each request, there is a credential in the header along with a verifier and authentication type. The responder can chose to authenticate the credential to confirm to its satisfaction that the credential is true. It is likely that the responder will use the authentication type and verifier to do the authentication. For example, for DES authentication, the credential might be a user name and the verifier is the user's password encrypted with a common key.
The RA (and therefore the BB) can chose to keep a list of credentials associated with each attribute as an access control list. The RA and BB can use the ACL to determine if an attribute is available for a particular request.

4.5 Access Control

Each attribute in the database has a list of users associated with it. These are the users that can know about that attribute. The responder must not return information about an attribute that the requester does not have access to. If there are no users associated with an attribute, then

anyone who has access to the database (see below) has access (i.e. can know about) that attribute.
The database itself also has an access control list. This list of users can have both read and write permissions. Database read permission means that a user can know about all the attributes in the database. Database write permission means that the user can modify the database (see *Database Add/Delete*).

4.6 Standard Attribute

4.6.1 Description

Although the protocol does not assume that the dictionary of attribute classes and values is standardized, it does not preclude standardization. For example, printer vendors might standardize on certain attribute classes and values to describe printers and ship resource agents and resource databases with their products.
This would allow vendors which supply client software (e.g. com software that uses modems, word processing software which uses printers) to generate the attribute expressions in resource requests. This avoids user interaction where the software knows what type of resource it needs and can use standard attributes.
The protocol specifies a way for telling the user agent that an attribute is standardized. Note that the attribute class, value, configuration string and help string must be standardized in order for the attribute to be considered standard. For example, if *type* is a standard attribute class, for *resource_type=printer* and *Helvetica* is a standard value, then the attribute *type=Helvetica* is a standard attribute (assuming the configuration and help strings are also standardized).
The RA database includes a field to indicate that an attribute is standardized. A UA can specify that specific *attr template*s in an attribute request may only match attributes in the RA database which are marked as standard. This guarantees that a UA that requires a standard attribute will get the semantics it desires and will not inadvertently conflict with a site-specific attribute which happens to syntactically match a standard attribute.
In the example above, an administrator may have configured his/her resources with the attributes *type=postscript* or *type=PCL5*. These attributes would be denoted as non-standard to avoid confusing them with the (hypothetical) standard attribute *type* which refers to a font.

4.6.2 Distinguished Attributes

Although the protocol does not limit the number of distinguished attributes, it is not currently feasible to have different distinguished attributes and maintain interoperability. A protocol between binding brokers connecting different administrative domains might allow this constraint to be relaxed. Until then, this RFC will standardize on the single distinguished attribute class *resource_type*.

4.6.2.1 Standard *resource_type* Values

The following are the legal values for the attribute class, *resource_type*:

- printer
- modem
- file_server
- name_server
- router
- mail_server

5 Protocol Data Units

5.1 Packet Header

All resource location packets begin with the following header

field	size (octets)	value
version	1	1 for now
function	1	LOCATE(1) RESOURCE (2) RESOURCE_DATABASE(3) DATABASE_ADD (4) DATABASE_DELETE(5)
length	2	bytes, data portion following header
xid	4	generated by sender, copied into response
error number	2	non-zero if error
flags	1	see explanation
authentication type	1	NO_CRED(0), NO_VERIFY(1), DES(2), KERBEROS(3)
resource database cookie	2	see explanation
sender's address	variable	*<address>*
credential	variable	*<string>*
verifier	variable	*<string>*

5.1.1 Header Flags

Value	Name	Description
0x01	response overflowed	Indicates that the response did not fit in a datagram. See *Connections* section

5.1.2 Resource Database Cookie

The magic cookie is a token passed back by the responder. It is an indication of the state of the responder's database. The cookie is opaque to the requester. All the requester knows is that when the responder's database is modified, the cookie will be changed. The requester should cache the cookie along with any other information which is cached in order to know when to request a cache update.
BBs and RAs must not use '0' for a cookie value.

5.2 PDU Data Portion

5.2.1 Locate

5.2.1.1 Description

Both BBs and RAs need to be located. There are three cases:

- Before the UA makes its first request, it tries to locate a BB, if it fails to locate a BB, it tries to locate multiple RAs

- When a BB comes up, it locates RAs to announce that it is up and the RAs need to register.
- When an RA comes up, it locates a BB to register with. The RA sends its database (using Database Transfer). It is the RA's responsibility to keep track of all BBs that respond to the locate request and to send its database to each.

5.2.1.2 Addressing

Each of the requests above can be either broadcast or multicast in accordance with the following protocol. The terms *should* and *may* are used as in RFC1122 and RFC1123.

- The requester should first multicast
- If the requester doesn't get an answer, then it may broadcast.
- The requester may cache the fact that it did not receive an answer to the multicast if it subsequently received an answer to its broadcast.
- The requester may assume that multicast is not implemented in its domain. The requester may then skip sending multicasts for some long period of time.

The broadcast packet is a modified bootp packet so that it can be passed across a router. The BBs and RAs must be able to accept bootp packets on the bootp port.

5.2.1.3 Silly Broadcast Algorithm

Because both broadcast and multicast are datagram based and create multiple responses, the protocol must address the issue of collecting multiple responses reliably. To find multiple entities:

1) set num_addresses = 0
1) broadcast the locate request
2) add the addresses of all respondents to the next locate request
3) Continue step #2 until the requester doesn't get any more answers

In addition to specifying which addresses you do not want to respond, you can also specify individual resources that you do not want to locate. So the algorithm that a binding broker and/or resource agent should use in deciding to respond to a locate request is:

Condition	**Response**
num addresses > 0	all resources
num addresses > 0 your address is not on the address list	all resources
num addresses > 0 your address is on the address list *num resources* > 0	any resources whose ids are not on the resources list, or whose database cookies are different than the one in the request

The requester can use the *locate* query to refresh its cache with new resource that have appeared on the network or existing resources whose database have changed. It puts the resource address that it knows about and database cookie in the request and broadcasts. Note that this will not tell it about RAs or BBs that have disappeared from the network. So the implementation should be robust about discarding cache entries that do not respond to subsequent requests.

5.2.1.4 UA Caching

UA may cache attribute information, specifically, it should maintain a cache mapping distinguished attributes to the address of the RA which supports that distinguished attribute. This cache information needs to timeout. The UA should timeout the entire cache. It can form a *locate* request with the RA address and resource id for each distinguished attribute in the cache. This means that the UA can update the cache with a single broadcast. RAs which receive the *locate* request should only respond if they have a resource which is not specified in the *locate* request or if the database cookie has changed. Note that this allows the UA to learn about new resources. However, it will not learn about resources or RAs which have disappeared from the network. Therefore, the UA must be robust if it doesn't get an answer from a resource. After a retry, the UA may chose to delete the distinguished attribute/address mapping from its cache. If this was a mistake or if the resource reappears on the net, the UA will learn about it the next time it updates its cache.

5.2.1.5 Distinguished Attributes

In addition to the address, the requester can request high-level resource information about the responder's resources. By setting the *DIST_ATTR* bit in *flags*, the requester asks the responders to includes a list of the distinguished attributes along with the resource ids.
There may be more than 1 distinguished attribute, but all resources in a domain must have the same number of distinguished attributes.

5.2.1.6 Scaling Issues

The *silly broadcast* algorithm scales until there are more resource agents than there is room in the locate request for the addresses and resource ids. Each IP address takes 7 octets plus one to indicate that we are not specifying resource ids (num resources = 0). The packet overhead is 16 octets (minimum) for the header and 5 octets for the fixed data portion. This means that a locate request can hold, at most, addresses for 69 RAs.
This broadcast re transmission algorithm is only necessary if the UA does not find a BB on the network. The solution for a network with more than 69 RAs is to install a BB.

5.2.1.7 RA Registration

When an RA comes up, it broadcasts a *locate* request with the responder set to *BB*. Any BB on the network must unicast a *locate* reply. *num distinguished attrs* and *num resource addresses* must be set to 0 in the locate reply. The RA then sends a *resource database* reply with its database. The BB can choose to retransmit if it doesn't receive a database from an RA that it received a broadcast from.
When the BB comes up, it broadcasts a *locate* request with the responder set to RA. Any RA which receives the broadcast must send a *locate* response with *num distinguished attrs* and *num resource addresses* set to 0. The BB sends a *resource database* request and the RA sends it database in the *resource database* reply. The RA must timeout and resend the *locate* response if it doesn't receive a *resource database* request. Since the BB broadcasts to a potentially large number of RAs, it might be flooded with *locate* responses. The RAs should backoff and retransmit to allow the BB to collect the addressees of all the RAs.

5.2.1.8 Request (Multicast)

field	size (octets)	data type or value
responder type	1/2	RA(2), BB(3)
requester type	1/2	UA(1), RA(2), BB(3)
requester port	2	
flags	1	DIST_ATTRS(0x1)
number of addresses	1	
for each address:		
resource address	variable	*<address>*
resource DB cookie	2	
num resources		
for each resource:		
resource id	2	

5.2.1.9 Request (Broadcast)

The broadcast-based request is a bootp request. There is a single option set:

option XXX (ResLoc Locate Request)
<XXX> <octets>

The format for <octets> is the ResLoc header followed by the data portion as specified for a multicast locate request.

5.2.1.10 Response

field	size (octets)	data type or value
responder type	1	RA(2), BB(3)
num distinguished attrs	1	
num resource addresses	2	
resource address	variable	*<address>*
resource DB cookie	2	
num resources	2	
for each resource		
resource id	2	
for each dist attr		
dist attr	variable	*<attribute>*

5.2.2 Resource Query

5.2.2.1 Description

The *resource* request contains a boolean expression of attributes which describes the characteristics of the resource that the UA wants. An RA or BB responds to a *resource* request to indicate that it has a resource whose attributes satisfy the boolean expression and responds with the list of attributes that make the boolean expression true. This allows the user to describe the resource that they need. The user then selects the resource which best meets their need by reviewing the attributes which satisfy the request

The UA puts a predicate in the packet and sends it to the RA or BB. The UA should cache the addresses and distinguished attributes found via the *locate* query. The UA

should scan the predicate in the *resource* query for distinguished attributes in order to determine which RA to send the resource request to. There are cases where the UA will be unable to limit the set of resources that should receive the request. The UA may then broadcast the *resource* request. However, since the distinguished attributes form a high-level taxonomy for resources, most *resource* requests will contain one and only one distinguished attribute which the UA should use for addressing.

For *resource* responses from a BB, there can be multiple resource addresses, corresponding to the various RAs registered at the BB. The response will specify the resource address of the RA which registered that resource at the BB.

5.2.2.2 Request

field	size (octets)	data type or value
predicate	variable	*<attr expr>*

5.2.2.3 Response

field	size (octets)	data type or value
num resource addresses	2	
for each resource address:		
resource address	variable	*<address>*
for each resource:		
resource id	2	
num matched attrs	2	
for each attribute	variable	*<attribute>*

5.2.3 Dictionary Query

5.2.3.1 Description

A requester uses the *dictionary* query to get the list of values for a particular attribute class.
A dictionary is similar to a list of attributes. Where a list of attributes is a list of the form *(class, value)*, a dictionary is of the form *(class, (set of values))*. The dictionary provides the terms necessary to build the predicate in a *resource* request.
The *dictionary* query can get the values for a single attribute at a single resource or get the entire dictionary for all resources at an RA or BB. A requester can get a dictionary for specific set of resources by setting the resource address (if sent to a BB) and the resource ids that should respond. If no resource addresses or ids are specified, all resources are selected.
The requester can also specify the attribute classes that it want the values for.
The *dictionary* response does not include any resource addresses or ids. The purpose of the *dictionary* query is not to address resources. The requester should take the dictionary it gets back and form a *resource* request to locate the resource.

5.2.3.2 Request

field	size (octets)	data type or value
num resource addresses	2	

for each address:		
num resource ids	2	
for each resource:	2	resource id
num attribute classes	2	
for each attribute class		
attribute class	variable	*<string value>*

5.2.3.3 Response

field	size (octets)	data type or value
num attribute classes	2	
for each attribute class:		
attribute class	variable	*<string value>*
enumerated type?	1	TRUE(1), FALSE(0)
data type	1	STRING(1), INT(2), ATTR_FUNC(3)
num attr values	2	
for each attribute value:		
standard attr?	1	TRUE(1), FALSE(0)
value	variable	*<integer value>* | *<string value>*

5.2.4 Resource Database

5.2.4.1 Description

The *resource database* query transfers information from the attribute database at an RA or BB. It is not used for locating resources. It is used after a resource has been located (resource address and id). Then the *resource database* query gets additional attribute information.

The requester specifies which attributes it want additional information for by specifying the class name and value for the attribute. The requester can specify a '*' as a value. The responder will then supply the specified fields for all attributes that have the specified class name.

The requester sets a bit mask to specify which fields the responder should return in the response. The fields are returned in the response in the order of the bits in the bit mask

5.2.4.2 Request

field	size (octets)	data type or value
num resource addresses	2	
for each address:		
num resource ids	2	
for each resource:	2	
resource id	2	
num attributes	2	
for each attribute:		
class name	variable	*<string>*
value	variable	*<integer value>* | *<string value>* |

		'*'
fields requested	2	ENUM?(0x01) STD_ATTR?(0x02) HELP_STR(0x04) CONFIG_STR(0x08) ACL(0x10)

5.2.4.3 Response

field	size (octets)	data type or value
num resource addresses	2	
for each address:		
num resource ids	2	
for each resource:	2	
resource id	2	
attribute	variable	*<attribute>*
enumeration	1	FALSE(0), TRUE(1)
standard attribute	1	FALSE(0), TRUE(1)
num info strings	2	
for each info string:	variable	*<string>*
num help strings	2	
for each help string:	variable	*<string>*
access control list	variable	*<acl>*

5.2.5 DB Admin (Database Add/Database Delete)

5.2.5.1 Description

These PDUs are used to remotely administer an RA's database. It pushes information into the database. It is not used for an RA to register with a BB. The registration happens when the BB requests a copy of the RA's database using the *resource database* PDU.

The database administration PDU is used to inject the resource information into the ResLoc system. It should be sent from an administration station to an RA since the RA has the authoritative information.

5.2.5.2 Request

The *Database Add* request uses the same PDU as the *Resource Database* response.

The *Database Delete* request uses the same PDU as the *Resource Database* request. Whatever database entries would be returned by the *Resource Database* request are deleted from the database.

5.2.5.3 Response

Neither the *Database Add* nor *Database Delete* responses have a data section. The relevant information (specifically, the error code) is in the header

6. Technical Issues

6.1 Resource Location without a User Interface

Attribute classes and values are meant to be self-describing strings and values. This assumes a user interface and human interpreter. However, it is possible for client software to use the

ResLoc protocol without a user interface. This requires a priori knowledge of the attribute classes and values in the administrative domain. This presupposes that either these dictionary terms have been standardized or that a vendor's client software is using a proprietary dictionary to locate its own resource. An example of this is a *gethostbyname* library which finds a DNS server by forming a resource request with the predicate:

resource_type = "DNS_server" *<standard attribute>*

Normally, to collect the information to form this request, the UA first sends a *locate* request to find the RA for the DNS server, followed by one or more *database get* requests (to learn about the attribute class *resource_type* and the value *DNS_server)*. If the UA already has these symbols, it can form the predicate without the *database get* request. If it broadcasts the *resource* request, it does not need the *locate* request.
Note that there is no provision for the UA being inundated with responses to a broadcast resource request as there is for *locate* requests. So the UA which broadcasts a resource request should be prepared to get answers from a subset of the resource agents.

6.2 Binding Broker to Binding Broker Protocol

UAs in one administrative domain can access resources in another administrative domain. The details of how two binding brokers share information will be specified in a future RFC. Until then, users will be unable to access RAs in other administrative domains. The administrative domain is defined by the bootp perimeter.

6.3 Connections

A requester may receive a response with the *response overflow* field set. This indicates that the responder couldn't fit all the information in a single datagram. It indicates that the responder is willing to establish a connection with the requester.

6.3.1 Choosing to implement connection-oriented

Only BBs are required to support connections. RAs and UAs may only support datagrams. This is in keeping with a desire to use RAs and UAs in embedded systems. Note that if the RA does not support connections, it should not set the *response overflow* field. This emphasizes the importance of the design decision to not implement connection-oriented in an RA. If the implementer does not implement connection-oriented, he/she must be able to guarantee that the packet will not overflow. The UA should not have to decide how to react to receiving an indication that it has partial information (i.e. response overflowed), yet no means to get the missing information. Therefore, the protocol puts the burden on the RA developer who has the information to make sure that either:

1. the information fits in a datagram
2. there is a means to get additional information (i.e. connection)
3. the RA will act as though the information which did fit is complete.

6.3.2 Negotiating the connection

The requester should resend the request on the well know connection-oriented port. The responder transmits the entire response, including the initial portion already sent. This wastes some bandwidth and processing at the responder. However, it means the responder doesn't have to keep an arbitrarily large amount of state and uses existing PDUs.

6.4 Resource Location over IP

IP networks have enough functions to support resource location. Addressing is done by setting the address type to 0x0800. Note that this is the same identifier in the DIX Ethernet type field. The address length is 4 for IPv4. The address contains the 4 octet IP address.
Broadcast is accomplished using IP broadcast. The protocol only requires the ability to broadcast on a single physical network. So the broadcast should not be an internet-wide broadcast (all 1's), but rather only the host portion should be set to 1's.
The requester can chose to use multicast to locate RAs and BBs.

6.5 Magic Numbers

The following are the fields which require *magic* numbers. The values defined in this document are repeated here for convenience.

6.5.1 IANA Registered Numbers

section(s)	description	value	meaning	IANA Registered?
	bootp port num			already done
6.4	address type	0x0800	IP	already done
6.4	RA port number			
6.4	BB port number			
5.2.1.2	multicast address			

6.5.2 Error Values

This section will be completed at a later date.

field	value	meaning
ENOERROR	0	All's well
ENOAUTH	1	The authentication information was rejected

6.5.3 Enumerations used in PDUs

section(s)	field	value	meaning
5.1	function	1	LOCATE
		2	RESOURCE
		3	RESOURCE_DATABASE
		4	DATABASE_ADD
		5	DATABASE_DELETE
5.1	*all booleans*	0	FALSE
5.2.3.3		1	TRUE
5.2.4.2			
5.1	authentication type	0	NO_CRED
		1	NO_VERIFY
		2	DES
		3	KERBEROS
5.2.1.7	type of process	1	UA
5.2.1.9		2	RA
		3	BB
5.2.3.2	data requested	1	ATTRS_ONLY
5.2.4.2		2	DICTIONARY
		3	ALL_ATTR_INFO

7 Glossary

Resource agent (RA)	A process which responds to queries with information about a resource.
User agent (UA)	The process (or library within a process) which forms network requests and receives the responses from the resource agents
Binding Broker (BB)	A process which collects information from, and proxies for, resource agents
Attribute	A description of some characteristic of a resource. An attribute is made up of an attribute class and an attribute value. The attribute class describes a category of attributes, e.g. speed, location, cost. The attribute value specifies the value for a given attribute class, e.g. location = room_8.
Standard Attribute	An attribute with a particular semantic that was defined *out-of-band* to the ResLoc protocol.
Distinguished Attribute	All resources must have at least one attribute class in common. This attribute class as well as its value is distinguished. To bootstrap a client that does not have any configuration, it first determines the distinguished attribute. Therefore, the distinguished attribute defines the high-level taxonomy for the resources.
Administrative Domain	All the RAs that register with a single BB. Since the RAs and UAs use bootp packets to locate a BB, the administrative domain is defined by the bootp radius.
Dictionary	All the terms (attribute classes and enumerated values) used to form attributes in a resource agent's database.
Requester	A UA, RA, or BB which sends a request PDU in order to get some information
Responder	A UA, RA, or BB which send back the answer to a request (i.e. a response)

8 Acknowledgments

Leo McLaughlin (FTP), Mike Ritter (Apple) and John Veizades (Apple) have all provided enormous technical contributions by reviewing both the design, implementation, and standardization of ResLoc. Bala Krishnaswamy (FTP), Nirmalkumar Samuel (FTP), and Derek Brown (FTP) took the initial design and shaped it into a working prototype. Nirmal and Ben Levy (FTP) reviewed the Internet Draft and produced the first implementation.

9 Author's Address

Scott Kaplan (scott@ftp.com)
FTP Software Inc., West Coast Operations
785 Market Street, 12th Floor
San Francisco, CA 94103
(415) 543-9001

10 References

M. Acetta. *Resource Location Protocol.* RFC 887, NIC, December 1983.

Legato Systems, *The Legato Resource Administration Platform,* Legato Systems, 1991.

C. McManis and R. Rom, *The Zeus Name Service Architecture,* Sun Microsystems, 1990.

S. Dyer, *The Hesiod Name Server,* Winter Usenix Conference, pp. 183-187, Feb 1988.

D. Oppen and Y. Dalal, *The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment,"* Tech. Rep. OPD-78103, Xerox Office Products Division, 1981.

B. Lampson, *Designing a Global Name Service,* Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, pp. 1-10, 1986.

D. Cheriton and T. Mann, *Uniform Access to Distributed Name Interpretations in the V-system.*

Sun Microsystems, *Remote Procedure Call Programming Guide, 1990.*

Sun Microsystems, *External Data Representation: Sun Technical Notes,* 1990.

R. Droms. *Dynamic Host Configuration Protocol*. RFC Draft, NIC, 1991

B. Croft and J. Gilmore. *Bootstrap Protocol (BOOTP)*. RFC 951, NIC, September 1985

P. Prindevill. *BOOTP Vendor Information Extensions.* RFC 1048, NIC February 1988.

J. Reynolds. *BOOTP Vendor Information Extensions.* RFC 1084, NIC, December 1988.

W. Wimer. *Clarifications and Extensions for the Bootstrap Protocol.* Internet Draft, NIC, 1991.

S. Deering. *Router Discover Protocol.* RFC 1256, NIC 1991.

P. Mockapetris. *Domain Names - Concepts and Facilities*. RFC 1034, NIC, November 1987

P. Mockapetris. *Domain Names - Implementation and Specification.* RFC 1035, NIC. November 1987

11 To Do List

1) How does a bootp gateway separate ResLoc requests from bootp requests?
2) We are hiding locates in bootp packets to deal with the case of multiple subnets with no binding broker. Is this a problem we need to solve?
3) Need to define <characters> in BNF
4) make the tables into **real** WinWord tables
5) generate ASCII text format
6) need port assignments for RA and BB in the *ResLoc over IP* section
7) put network architecture diagram in *Overview* section
8) get more examples for distinguished attribute values (section 6.1.3) from Mockapetris RFC for *WKS* values
9) What do we use for address type?
10) Is READ permission on attributes and READ/WRITE permission on the database sufficient access control?
11) What's a user?