JXTA v1.0 Protocols Specification

Revision 1.1.1
June 12, 2001

discuss@jxta.org

Table of Contents

Introduction 1

The JXTA Protocols

The JXTA protocols are a set of six protocols that have been specifically designed for *ad hoc*, pervasive, and multi-hop peer-to-peer (P2P) network computing. Using the JXTA protocols, peers can cooperate to form self-organized and self-configured peer groups independently of their positions in the network (edges, firewalls), and without the need of a centralized management infrastructure.

In designing the JXTA protocols, we sought to create a set of protocols that had very low overhead, made few assumptions about the underlying network transport and limited requirements of the peer environment, but yet were able to be used to deploy a wide variety of P2P applications and services in a highly unreliable and changing network environment.

Peers use the JXTA protocols to advertise their resources and to discover network resources (services, pipes, etc.) available from other peers. Peers form and join peergroups to create special relationships. Peers cooperate to route messages allowing for full peer connectivity. The JXTA protocols allow peers to communicate without needing to understand or manage the potentially complex and dynamic network topologies which are becoming common.

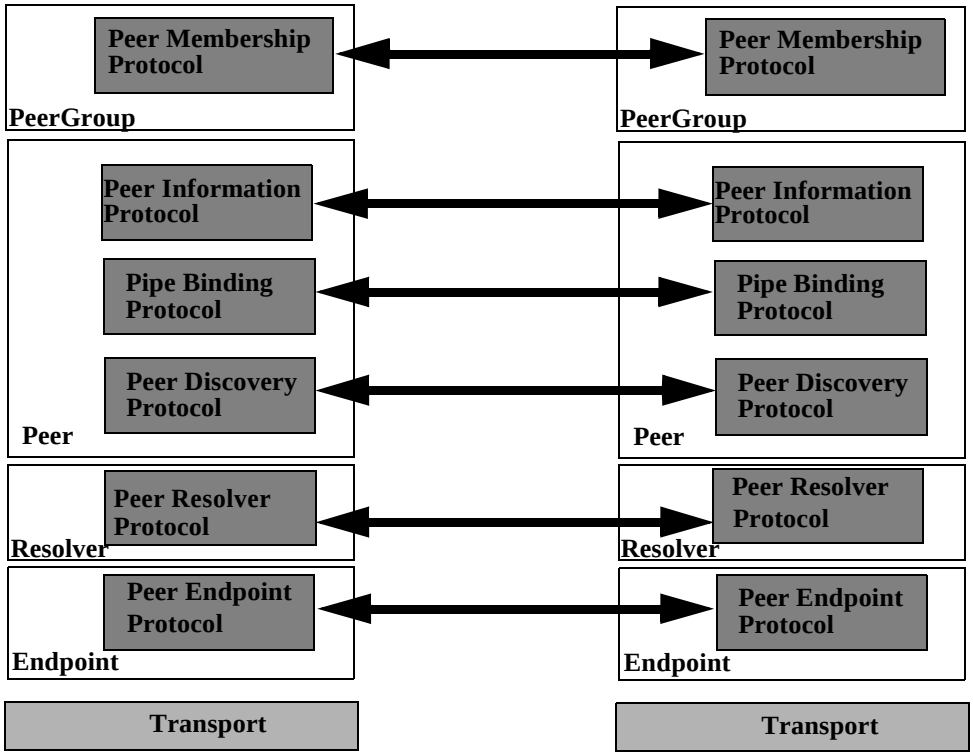
The JXTA protocols allow peers to dynamically route messages across multiple network hops to any destination in the network (potentially traversing firewalls). Each message carries with it either a complete or partial ordered list of gateway peers through which the message might be routed. If a route information is incorrect, the intermediate peer can assist in dynamically finding a new route.

The JXTA protocols are composed of six protocols that work together to allow the discovery, organization, monitoring and communication between peers:

- The **Peer Discovery Protocol** (PDP) is the mechanism by which a peer can advertise its own resources, and discover the resources from other peers (peer groups, services, pipes and additional peers). Every peer resource is described and published using an *advertisement*. Advertisements are programming language-neutral metadata structures that describes network resources. Advertisements are represented as XML documents.
- The **Peer Resolver Protocol** (PRP) is the mechanism by which a peer can send a query to one or more peers, and receive a response (or multiple responses) to the query. The PRP implements a query/response protocol. The response message is matched to the query via a unique id included in the message body. When a peer is discovered via PDP, a query can be sent to that peer.
- The **Peer Information Protocol** (PIP) is the mechanism by a which a peer may obtain status information about other peers, such as state, uptime, traffic load, capabilities, etc.
- The **Peer Membership Protocol** (PMP) is the mechanism by which peers can self-organize and group themselves into peer groups. Peer groups form a logical boundary of peers with a common interest. A single peer can belong to multiple peer groups. PMP is used by a peer to join or leave an existing peergroup discovered via the PDP.
- The **Pipe Binding Protocol** (PBP) is the mechanism by which a peer can establish a virtual communication channel or *pipe* between one or more peers. The PBP is used by a peer to bind the two or more ends of the connection (pipe endpoints). Pipes provide the foundation communication mechanism between peers.
- The **Peer Endpoint Protocol** (PEP) is the mechanism by which a peer can discover a route (sequence of hops) used to send a message to another peer. If a peer A wants to send a message to peer C, and there is no direct route between A and C, then peer A needs to find the intermediary peer(s) to route the message to C. PEP is used to determine the route information. If the network topology has changed such that the route to C can no longer be used, because a link along the route no longer works, the peer can use PEP to find any routes other peers know to construct a route to C.

All of these protocols are implemented using a common messaging layer. This messaging layer is what binds the JXTA protocols to various network transports. (See “Messages”)

Each of the JXTA protocols is independent of the others, and a peer is **NOT REQUIRED** to implement all six protocols. A peer just needs to implement the protocols that it needs to use. For example, a device may have all the advertisements it uses pre-stored in memory, so that peer does not need to implement the Peer Discovery Protocol. A peer may use a pre-configured set of peer routers to route all its messages, hence the peer does not need to implement the Peer Endpoint protocol. It just sends messages to the routers to be forwarded. A peer may not need to obtain or wish to provide status information to other peers, hence the peer does not to implement the Peer Monitoring Protocol. The same can be said about all of the other protocols.



JXTA Protocols

The JXTA protocols do no require periodic messages of any kind at any level to be sent within the network. For example, JXTA does not require periodic polling, link status sensing, or neighbor detection messages, and does not rely on these functions

from any underlying network transport in the network. This entirely on-demand behavior of the JXTA protocols and lack of periodic activity allows the number of overhead messages caused by JXTA to scale all the way down to zero, when all peers are stationary with respect to each other and all routes needed for current communication have already been discovered.

A peer may decide to cache advertisements discovered via the Peer Discovery Protocol for later usage. It is important to point out that caching is not required by the JXTA architecture, but caching can be an important optimization. The caching of advertisements by a peers avoids performing a new discovery each time the peer is accessing a network resource. In highly-transient environment, performing the discovery is the only viable solution. In static environment, caching is more efficient.

An unique characteristic of P2P networks, like JXTA, is their ability to replicate information toward end-users. Popular contents tend to be replicated more often, making them easier to find as more copies are available. Peers do not have to always go back to the same peer to obtain the information they want (current client/server model). Peers can obtain information from neighboring peers that have already cached the information. Each peer becomes a provider to all other peers.

The JXTA protocols have been designed to allow JXTA to be easily implemented on uni-directional links and asymmetric transports. In particular, many forms of wireless networking do not provide equal capability for devices to send and receive. JXTA permits any uni-directional link to be used when necessary, improving overall performance and network connectivity in the system. The intent is for the JXTA protocols to be as pervasive as possible, and easy to implement on any transport. Implementations on reliable and bi-directional transports such as TCP/IP or HTTP should lead to efficient bi-directional communications.

The JXTA uni-directional and asymmetric transport also plays well in multi-hops network environment where the message latency may be difficult to predict. Furthermore, peers in a P2P network tends to have nondeterministic behaviors. They may appear or leave the network very frequently

JXTA Assumptions

This section is a guide to the assumptions that inform the design of JXTA. There are two types of assumptions stated here, those which describe the requirements of JXTA implementations and those which describe the expected behavior of the JXTA network.

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in IETF RFC 2119. (HTTP://IETF.ORG/RFC/RFC2119.TXT)

A peer **SHALL NOT** make assumptions about the runtime environments or programming languages in use by another peer. The network of peers reachable by any peer is likely to be contain many peers with very heterogeneous implementations and capabilities.

You **SHOULD** assume that capability and complexity of the network peers supporting these protocols can range from a single light switch to a highly-available supercomputer cluster.

A peer **MUST** implement the JXTA protocols such that all interaction with other peers is correct and conformant.

A peer **MAY** implement only the JXTA protocols it requires and for correct and conformant interaction with other peers.

A peer **MAY** implement only a portion (client-side or server-side only for example) of a protocol.

Peers wishing to interact with other peers within the JXTA network **SHOULD** be willing to participate fully in the protocols. In particular, peers **SHOULD** cache advertisements and forward messages for other peers in the JXTA network. But, this participation is **OPTIONAL**.

The JXTA protocols **MAY** be implemented over the Internet, a corporate intranet, a dynamic proximity network, or in a home networking environment.

Peers receiving a corrupted or compromised message **MUST** discard the message. Messages may be corrupted or intentionally altered in transmission on the network. The JXTA protocols provide

Peers **MAY** appear, disappear and move at any time without notice. In particular, the JXTA protocols support very arbitrary environment changes allowing a peer to dynamically discover and reconnect to its changing environment.

Communication ability between any pair of peers **MAY** at times not work equally well in both directions. That is, communications between two peers will in many cases be able to operate bi-directionally, but at times the connection between two peers may be only uni-directional, allowing one peer to successfully send messages to the other while no communication is possible in the reverse direction.

The JXTA protocols **MAY** take advantage of additional optimizations, such as the easy ability to reverse a source route to obtain a route back to the origin of the original route.

The JXTA protocols are defined as idempotent protocol exchanges. The same messages **MAY** be sent/received more than once during the course of a protocol exchange. No protocol states are REQUIRED to be maintained at both ends.

Due to unpredictability of P2P networks, assumptions **MUST NOT** be made about the time required for a message to reach a destination peer. JXTA Protocols **SHALL NOT** impose any timing requirements for message receipt.

A JXTA protocol message which is routed through multiple hops **SHOULD NOT** be assumed to reliably delivered, even if only reliable transports such as TCP/IP are used through all hops. A congested peer **MAY** drop messages at any time rather than routing them.

Bidirectional, reliable data transfers or specific data transfers (streaming) are expected to be provided at a service layer.

Message encodings for network transports **MUST** allow for the transmission of arbitrary numbers of message sections containing an arbitrary amount of data. All data should be

While the JXTA protocols are defined in term of XML messages, an XML parser is **OPTIONAL**. XML messages **MAY** be pre-compiled on small implementations.

The diameter of a multi-hops network is the minimum number of hops necessary for a message to reach from any peers located at one extreme edge of the network to another peer located at the opposite extreme. Empirical measurements on P2P networks such as Gnutella or Freenet shows this diameter to be around 5-7 hops.

The JXTA protocols **MUST NOT** require a broadcast or multicast capability of the underlying network transport. Messages intended for receipt by multiple peers (propagation) **MAY** be implemented using point-to-point communications.

One **SHOULD** make the assumption that if a destination address is not available at any time during the message transmission, the message will be lost.

Each peer **MUST** be a member of the World peer group. Membership in this group is automatic.

Peers **MUST** be members of the same peer group in order to exchange messages. (Remember that all peers are members of the World peer group)

A peer **MUST NOT** assume that there is a guaranteed return route to a peer from which it has received communication. The lack of a return route may either be temporary or permanent.

Names are not unique unless a coordinated naming service is used to guarantee name uniqueness. A naming service is **OPTIONAL**.

Once a content has been published to the JXTA network, it **MUST NOT** be assumed that that the content can be later retrieved from the JXTA network. The content may be only available from peers that are not currently reachable or nowhere.

Once a content has been published to the JXTA network, it **MUST NOT** be assumed that that the content can be deleted. Republication of content by peers is unrestricted and the content may propagate to peers which are not reachable from the publishing peer.

Why JXTA?

The JXTA Project is an open network computing platform designed for peer-to-peer (P2P) computing.

The JXTA protocols standardize the manner in which peers:

- Discover each others
- Self-organize into peer groups
- Advertise and discover network resources
- Communicate with each others
- Monitor each other

The JXTA protocols do **NOT**:

- Require the use of any particular computer language or operating system.
- Require the use of any particular network transport or topology.
- Require the use of any particular authentication, security or encryption model.

The JXTA protocols enable developers to build and deploy interoperable services and applications, further spring-boarding the peer to peer revolution on the Internet.

The JXTA Project intends to address this problem by providing a simple and generic P2P platform to host any kinds of network services:

- JXTA is defined by a small number of protocols. Each protocol is easy to implement and integrate into P2P services and applications. Thus service offerings

from one vendor can be used transparently by the user community of another vendor's system.

- The JXTA protocols are defined to be independent of programming languages, so that they can be implemented in C/C++, Java, Perl, and numerous other languages. Heterogeneous devices with completely different software stacks can interoperate with the JXTA protocols.
- The JXTA protocols are designed to be independent of transport protocols. They can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA, and many other protocols.

The JXTA Three Layer Cake

The JXTA Project is divided in three layers.

- **Platform**. This layer encapsulates minimal and essential primitives that are common to P2P networking, including peers, peergroups, discovery, communication, monitoring, and associated security primitives. This layer is ideally shared by all P2P devices so that interoperability becomes possible.
- **Services**. This layer includes network services that may not be absolutely necessary for a P2P network to operate but are common or desirable to be available to the P2P environment. Examples of network services, include search and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication and PKI services.
- **Applications**. This layer includes P2P instant messaging, entertainment content management and delivery, P2P E-mail systems, distributed auction systems, and many others. Obviously, the boundary between services and applications is not rigid. An application to one customer can be viewed as a service to another customer.

Conceptual Overview 2

JXTA is intended to be a small system with a limited number of concepts at its core. This chapter introduces the concepts which are core to JXTA.

Peers

A *peer* is any networked device (sensors, phones, PDAs, PCs, servers, supercomputers, etc.) that implements one or more of the JXTA protocols.

Each peer operates independently and asynchronously of all other peers. Some peers **MAY** have more dependencies with other peers due to special relationships (gateways or routers).

Peers spontaneously discover each other on the network to form transient or persistent relationships called *peer groups*. Peergroups are a collection of peers that have some common interests. Peergroups **MAY** also be statically predefined.

Peers of the same kinds tend to be inter-changeable. It **MAY** not matter which peers a peer interact with.

Peers **MAY** publish network resources (CPU, Storage, Routing) to other peers.

A peer **MAY** cache information, but it is **OPTIONAL**. Peers **MAY** have persistent storage.

Peers typically interact with a small number of other peers (network neighbors or buddy peers).

Assumptions **MUST NOT** be made about peer reliability or connectivity. A peer **MAY** appear or leave the network at any time.

Peers **MAY** provide network services that can be used by other peers.

Peers **MAY** have multiple network interfaces, though a peer does not need to publish all of its interfaces for use with the JXTA protocols. Each published interface is advertised as a *peer endpoint.* A peer endpoint is a URI that uniquely identify a peer network interface. Peer endpoints are used by peers to establish direct point-to-point connection between two peers.

Peers may not have direct point-to-point network connection between themselves, either due to lack of physical network connections, or network configuration (NATs, firewalls, proxies, etc.). A peer **MAY** have to use one or more intermediary peers to route a message to another peer.

Each peer is uniquely identified by a unique Peer Id.

Peer Groups

Peers self-organize into *Peer Groups*. A peer group is a collection of peers that have a common set of interests. Each peer group is uniquely identified by a unique PeerGroup Id. The JXTA protocols does not dictate when, where, or why peergroups are created. The JXTA protocols only describe how a peers may publish, discover, join, and monitor peergroups.

JXTA recognizes *three common motivations* for creating peer groups:

- To create a secure environment. Peergroup boundaries permit member peers to access and publish protected contents. Peergroups form logical regions which boundaries limit access to the peergroup resources. A peergroup does not necessarily reflect the underlying physical network boundaries such as those imposed by routers and firewalls. Peergroups virtualize the notion of routers and firewalls, subdividing the network in secure regions without respect to actual physical network boundaries.
- To create a scoping environment. Peergroups are typically formed and self-organized based upon the mutual interest of peers. No particular rules are imposed on the way peergroups are formed, but peers with the same interests will tend to join the same peergroups. Peergroups serve to subdivide the network into abstract regions providing an implicit scoping mechanism. Peergroup boundaries define the search scope when searching for a group’s content.
- To create a monitoring environment. Peergroups permit peers to monitor a set of peers for any special purpose (heartbeat, traffic introspection, accountability, etc.).

A peergroup provides a set of services called *peergroup services*. JXTA defines a core set of peergroup services. The JXTA protocols specify the wire format for these core peergroup services. Additional peergroup services can be developed for delivering specific services. For example, a lookup service could be implemented to find *active* (running on some peer) and *inactive* (not yet running) service instances. The core peer group services are:

- Discovery Service
  - The Discovery service is used by a peer members to search for peergroup resources (peers, peer groups, pipes).
- Membership Service
  - The membership service is used by the current members to reject or accept a new group membership application.
  - We expect that most peergroups will have at least a membership service, though it may be a “null” authenticator service which imposes no real membership policy. The membership service is used by a member peer to allow a new peer to join a peergroup. In order for a peer to join a peergroup, a peer is **REQUIRED** to have discovered at least one member of the peergroup.
  - Peers wishing to join a peer group must first locate a current member, and then request to join. The application to join is either rejected or accepted by the collective set of current members. The membership service **MAY** enforce a vote of peers or elect a designated group representative to accept or reject new membership applications. A peer **MAY** belong to more than one peergroup simultaneously.
- Access Service
  - The Access service is used to validate requests made by one peer to another. The peer receiving the request provides the requesting peers credentials and information about the request being made to the Access Service to determine if the access is permitted.
  - Not all actions within the peer group need to be checked with the Access Service, only those actions which only some peers are permitted to use.
- Pipe Service
  - The pipe service is used to manage and create pipe connection between the different peer group members.
- Resolver Service

- The resolver service is used to send query string to peers to find information about a peer, a peer group, a service or a pipe.

- Monitoring Service
  - The monitoring service is used to allow one peer to monitor other members of the same peer group.

Not all the above services **MUST** be implemented by a peergroup. Each service has a defined protocol, the specifications for which are the main content of this document.

Network Services

Peers cooperate and communicate to publish, discover and invoke *network services*. A peer can publish as many services that it can provide. Peers discover network services via the Peer Discovery Protocol.

The way network services are invoked is beyond the scope of this specification. Upcoming standards such as WSDL, ebXML, SOAP, UPnP can be used.

The JXTA protocols recognize two *levels* of network services:

- Peer Services
- PeerGroup Services

A peer service is accessible only on the peer that is publishing the service. If that peer happens to fail, then service also fails. Multiple instances of the service can be run on different peers, but each instance publishes its own advertisement.

A peergroup service is composed of a collection of instances (potentially cooperating with each other) of the service running on multiple members of the peergroup. If any one peer fails, the collective peergroup service is not affected, because chances are the service is still available from another peer member. Peergroup services are published as part of the peergroup advertisement.

Services can either be pre-installed into a peer or loaded from the network. The process of finding, downloading and installing a service from the network is similar to performing a search on the internet for a web page, retrieving the page, and then installing the required plug-in. In order to actually run a service, a peer may have to locate an implementation suitable for the peer's runtime environment. Multiple implementations of the same service may allow Java peers to use Java code implementations, and native peers to use native code implementations.

Service Invocation

Service invocation is beyond the scope of JXTA. JXTA intent is to interoperate and be compatible with any Web service standard (WSDL, uPnP, RMI, etc.). The JXTA protocols define a generic framework to publish and discover any kinds of service advertisements. Peers publish and discover service advertisements via the Peer Discovery Protocol. The service advertisement will typically contain all the necessary information to invoke the service. The JXTA protocols define a Service Advertisement, but it is for the most part open ended and allows for any form of service specification.

JXTA-Enabled Service

JXTA-Enabled services are services that publish pipe advertisements as their main invocation mechanism. The service advertisement specifies one or more pipe advertisements that can be used by a peer to create output pipes to invoke the service. The service advertisement also contains a list of pre-determined messages that can be sent by a peer to interact with the service. The service advertisement describes all messages that a client may send or receive. JXTA-Enabled service advertisements extend the W3C WSDL service schema where the service port is described as a pipe advertisement. Once a service is discovered, pipes can be used to communicate with the service.

Each service has a unique id that uniquely identifies the service.

Pipes

Pipes are virtual communication channels used to send and receive messages between services or applications over two peer endpoints. Pipes provide a network abstraction over the peer endpoint transport. Peer endpoints correspond to the available peer network interfaces that can be used to send and receive data from another peer. Pipes provide the illusion of a virtual in and out mailbox that is independent of any single peer location, and network topology (multi-hops route).

Different quality of services can be implemented by a pipe. For example:

- Uni-directional asynchronous: The endpoint sends a message, no guarantee of delivery is made.
- Synchronous request-response: The endpoint sends a message, and receives a correlated answer.
- Bulk transfer: Bulk reliable data transfer of binary data.

- Streaming: Efficient control-flow data transfer.

The uni-directional and asynchronous pipe is the **REQUIRED** by the JXTA protocols, the other forms may be used my other services and protocols.

Pipes connect one or more peer endpoints. At each endpoint, software to send, or receive, as well as to manage associated pipe message queues or streams is assumed, but message queues are **OPTIONAL**. The pipe endpoints are referred to as input pipes (receiving end) and output pipes (sending end).

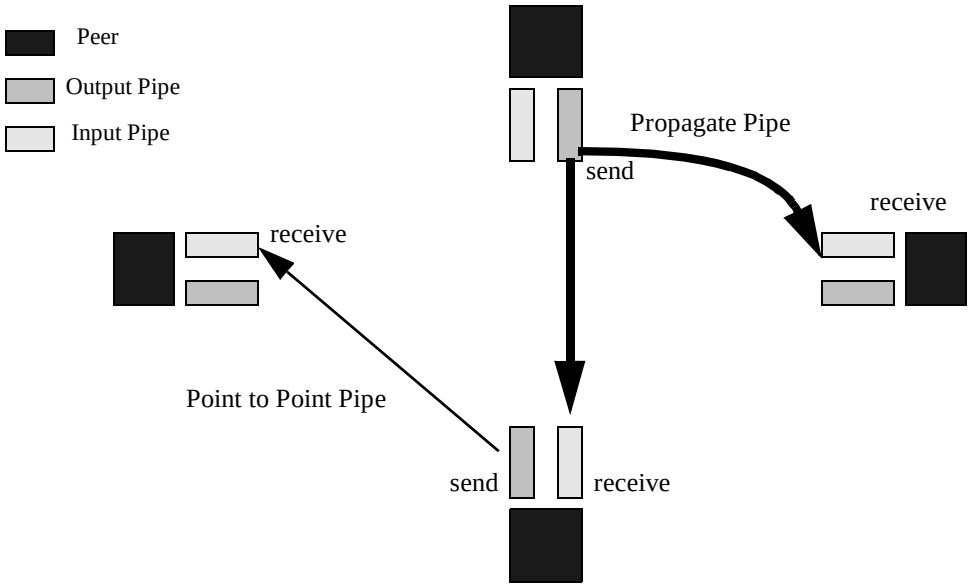
Pipe endpoints are dynamically bounded to a peer endpoint at runtime, via the *Pipe Binding Protocol* (see Protocols chapter). The pipe binding process consists of searching and connecting the two or more endpoints of a pipe.

When a message is sent into a pipe, the message is sent by the local output pipe to the destination pipe input currently listening to this pipe. The set of currently listening peer endpoints (where the input pipe is located) is discovered using the *Pipe Binding Protocol*.

A pipe offers two modes of communication:

- Point to Point
  - A point to point pipe connects *exactly two* pipe endpoints together, an input pipe that receives messages sent from the output pipe. No reply or acknowledgement operation is supported. Additional information in the message payload like a unique id may be required to thread message sequences. The message payload may also contain a pipe advertisement that can be used to open a pipe to reply to the sender (send/response).
- Propagate Pipe
  - A propagate pipe connects one output pipe to multiple input pipes together. Messages flow into the input pipes from the output pipe (propagation source). A propagate message is sent to all listening input pipes. This process may create multiple copies of the message to be sent. On TCP/IP, when the propagate scope maps an underlying physical subnet in a 1 to 1 fashion, IP multicast may be used as an implementation for propagate. Propagate can be implemented using point to point communication on transports that do not provide multicast such as HTTP.

Figure 2-1 Pipe Modes



Pipes may connect two peers that do not have a direct physical link. One of more intermediary peer endpoints are used to route messages between the two pipe endpoints

Messages

The information transmitted using pipes is packaged as messages. Messages define an XML envelope to transfer any kinds of data. A message **MAY** contain an arbitrary number of named sub-sections which can hold any form of data.

It is the intent that the JXTA protocols be compliant with W3C XML Protocol standards, so the JXTA protocols can be implemented on XML transports such as SOAP, XML-RPC, etc.

Binary data may be encoded using Base64 encoding scheme in the body of an XML message. A CDATA section may also be used.

The JXTA protocols are specified as a set of XML messages exchanged between peers. Each software platform binding describes how a message is converted to and from a native data structures such as a Java object or ‘C’ structure.

The use of XML messages to define protocols allows many different kinds of peers to participate in a protocol. Each peer is free to implement the protocol in a manner best suited to its abilities and role.

Advertisements

All network resources, such as peers, peergroups, pipes and services are represented by an *advertisement*. Advertisements are JXTA’s language neutral metadata structures that describe JXTA related peer resources. The JXTA protocols define the following advertisement types:

- Peer Advertisement.
- PeerGroup Advertisement
- Pipe Advertisement.
- Service Advertisement.
- Content Advertisement.
- Endpoint Advertisement.

The complete specification of advertisements is given in the Advertisements chapter. The JXTA protocols chapters make heavy reference to advertisements, so the reader should be familiar with advertisements before moving on the protocol specification chapters. Advertisements are by far the most common document exchanged in the protocol messages.

Services or peer implementations MAY subtype any of the above advertisements to create their own advertisements.

Credentials

The need to support different levels of resource access in a dynamic and ad hoc P2P network leads to a role-based trust model in which an individual peer will act under the authority granted to it by another trusted peer to perform a particular task. Peer relationships MAY change quickly and the policies governing access control need to be flexible in allowing or denying access.

Four basic security requirements MUST be provided:

- Confidentiality: guarantees that the contents of the message are not disclosed to unauthorized individuals.
- Authorization: guarantees that the sender is authorized to send a message.
- Data integrity: guarantees that the message was not modified accidentally or deliberately in transit.
- Refutability: guarantees the message was transmitted by a properly identified sender and is not a replay of a previously transmitted message.

XML messages allow to add a large variety of metadata information to message such as credentials, digests, certificates, public keys, etc.

A credential is a token that when presented in a message body is used to identify a sender and can be used to verify that sender's right to send the message to the specified endpoint. The credential is an opaque token that must be presented each time a message is sent.

The sending address placed in the message envelope is cross-checked with the sender's identity in the credential. Each credential's implementation is specified as a plug-in configuration, which allows multiple authentication configurations to co-exist on the same network.

Message digests guarantee the data integrity of messages. Messages may also be encrypted and signed for confidentiality and refutability.

It is the intent of the JXTA protocols to be compatible with widely accepted transport-layer security mechanisms for message-based architectures such as Secure Sockets Layer (SSL) and Internet Protocol Security (IPSec). However, secure transport protocols such as SSL and IPSec only provide the integrity and confidentiality of message transfer between two communicating peers. In order to provide secure transfer in multi-hops network, one MUST establish a trust association among all the intermediaries peers. Security is compromised if anyone of the communication links is not secured.

IDs

Within the JXTA protocols there are a number of entities that need to be uniquely identifiable. These are peers, peergroups, pipes and contents. A JXTA ID uniquely identifies an entity and serves as a canonical way of referring to that entity.

URNs are used for the expression of JXTA IDs.

Content

The JXTA protocols assume that many kinds of contents may be shared, exchanged, and replicated between peers. A content can be a text file, a structured document (like a PDF or a XML file), a Java ".jar" or loadable library, code or even an executable process (checkpointed state). No size limitation is assumed.

A content is published and shared amongst the peer members of a peergroup. A content MAY only belong to one peergroup. If the same content must be published in two different peergroups, two different contents **MUST** be created.

Each content is uniquely identified by a unique id.

All contents make their existence known to other members by publishing a content advertisement.

An instance of content is a copy of a content. Each content copy may be replicated on different peers in the peergroup. Each copy has the same content id as well as a similar value.

Replicating contents within a peergroup help any single item of content be more available. For example, if an item has two instances residing on two different peers, only one of the peers needs to be alive and respond to the content request.

The JXTA protocols does not specify how contents are replicated. This decision is left to a higher-level content service manager.

IDs 3

Introduction

Within the JXTA protocols there are a number of entities that need to be uniquely identifiable. Currently these are peers, peer groups, pipes and contents. In the future additional types of entities may be defined. A JXTA ID uniquely identifies some entity and serves as a canonical way of referring to that entity.

Originally a URI type was specified for JXTA IDs, “jxta:”. After further consideration it seems that URNs are a better choice for the expression of JXTA IDs. The “jxta:” URI form will be used for other types of references within JXTA, but no longer for the JXTA IDs. See IETF RFC 2141 for more information on URNs.

Format of a JXTA URN

A JXTA URN is not a type of URI, but rather a namespace for a URN. The namespace identifier “jxta” identifies a JXTA URN. Following the namespace is a textual encoding of a JXTA ID. The format allows for a wide variety of possible JXTA ID representations and constructions. Common to all representations is that the last two characters of the representation contain an identifier for the encoding format. Currently two encoding formats have been defined.

```
<JXTAURN>      ::= "urn:jxta:"1*<URN chars> <JXTA encoding>

<JXTA encoding> ::= <hex> <hex>

<URN chars>    ::= <trans> | "%" <hex> <hex>

<trans>       ::= <upper> | <lower> | <number> | <other> |
```

```
                    <reserved>
<upper>       ::=  "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
                   "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
                   "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" |
                    "Y" | "Z"
<lower>        ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                  "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                  "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
                   "y" | "z"
<number>       ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                   "8" | "9"
<hex>          ::= <number> | "A" | "B" | "C" | "D" | "E" | "F" |
                   "a" | "b" | "c" | "d" | "e" | "f"
<other>        ::= "(" | ")" | "+" | "," | "-" | "." |
                   ":" | "=" | "@" | ";" | "$" |
                   "_" | "!" | "*" | "'"
<reserved>    ::= "%" | "/" | "?" | "#"
```

The jxta URN namespace does not currently define any special symbols from the reserved set.

Format of the JXTA NULLID URN

There are two special reserved JXTA IDs. These are the NULL id and the World Peer Group ID. All implementations must support this ID encoding.

```
<JXTANULLURN> ::= "urn:jxta:" <JXTANULLENCODINGID> "-"
                  <JXTANULLENCODING>
<JXTANULLENCODING> ::= "00"
<JXTANULLENCODINGID> ::= <JXTANULL> | <JXTAWORLDGROUP>
<JXTANULL>     ::= "null"
<JXTAWORLDGROUP> ::= "worldgroup"
```

Format of a JXTA UUID URN

The initial implementation of JXTA IDs uses UUIDs encoded as hex digits as basis for the uniqueness of the identifiers. This ID scheme is identified by “01” as the final two hex characters of the URN. Immediately preceding this identifier are two hex characters that identify the type of JXTA ID. Currently, four ID Types are defined. Preceding the ID Type is the data portion of the ID.

```
<JXTAUUIDURN> ::= "urn:jxta:"(1*(<hex> <hex>)) <JXTAUUIDIDTYPE>
                    <JXTAUUIDENCODING>
<JXTAUUIDENCODING> ::= "01"
<JXTAUUIDIDTYPE> ::= <CODATID> | <PEERGROUPID> | <PEERID> |
                       <PIPEID>
<CODATID>      ::= "01"
<PEERGROUPID>  ::= "02"
<PEERID>       ::= "03"
<PIPEID>       ::= "04"
```

The characters preceding the ID type identifier are the encoded form of the ID. The encoding consists of a variable number of characters dependant upon the ID type being encoded. To decode the ID the hex characters are translated in order into the elements of a 64-byte array from which the various ID components can be retrieved. All of JXTA UUID IDs are manipulated as 64 byte arrays though no ID type currently requires all the full 64 bytes to encode their contents. Position 63 always contains the UUID Encoding value “01” and position 62 contains the UUID ID Type value. The remainder of the ID fields are defined beginning at Position 0 and increment towards Position 62.

To make the text encoding of JXTA IDs as URNs more compact implementations must not encode the value of unused Positions within the array. Since they are irrelevant to the value of the ID they can assumed to be zero. Implementations must also omit from the encoding the value of any Positions that precede the unused portion and contain zero. The reference Java implementation accomplishes this by scanning from Position 61 towards Position 0 searching for the first non-zero value. It then encodes from position 0 to the discovered location followed by the encoding for Positions 62 and 63. The text encoding of a JXTA ID must be canonical according to the URN specification, thus this “zero-saving” technique must be present in every implementation.

Example:

```
"urn:jxta:000301020450201"
```

Decodes to:

0:00	1:03	2:01	3:20	4:45	5-61:00	62:02	63:01

JXTA UUID Field Definitions

Each of the four types of JXTA IDs has a specific definition for how its fields are represented within the common 64-byte array structure. Common between the four types is the definition of Positions 62 and 63. These are reserved, respectively for the ID Type and Encoding fields.

JXTA UUID CodatID Fields

0:Group MSB	...GROUP UUID...						
...GROUP UUID (cont.)...							15:Group LSB
16:ID MSB	...CODAT UUID...						
...CODAT UUID (cont.)...							31:ID LSB
32:Hash0	CODAT SHA1 Hash...						
...CODAT SHA1 Hash...							
...CODAT SHA1 Hash			51:Hash19				
						62:IDType	63:Encoding

JXTA UUID PeerGroupID Fields

0:MSB	...GROUP UUID...	
...GROUP UUID (cont.)		15:LSB
	62:IDType	63:Encoding

JXTA UUID PeerID Fields

0:Group MSB	...GROUP UUID...	
...GROUP UUID (cont.)...		15:Group LSB
16:ID MSB	...PEER UUID...	
...PEER UUID (cont.)...		31:ID LSB
	62:IDType	63:Encoding

JXTA UUID PipeID Fields

0:Group MSB	...GROUP UUID...	
...GROUP UUID (cont.)...		15:Group LSB
16:ID MSB	...PIPE UUID...	
... PIPE UUID (cont.)...		31:ID LSB
	62:IDType	63:Encoding

Advertisements *4*

Introduction

The JXTA protocols use advertisements to describe, and publish the existence of a peer resources.

The JXTA protocols define the following advertisements:

- Peer Advertisement.
- PeerGroup Advertisement
- Pipe Advertisement
- Service Advertisement
- Content Advertisement
- Endpoint Advertisement

Advertisements are represented as XML documents. Users may subtype these advertisements to create their own types. Subtypes may add an unlimited amount of extra and richer metadata information.

XML

All JXTA advertisements are represented in XML. XML provides a powerful means of representing data and metadata throughout a distributed system. XML provides universal (software-platform neutral) representation:

- XML is language agnostic.

- XML is self-describing.
- XML is strongly-typed.
- XML ensures correct syntax.

All XML advertisements are strongly typed and validated using XML Schemas. Only well-formed XML documents that descend from the base XML advertisement types are accepted by peers supporting the various protocols.

The other powerful feature of XML is its ability to be translated into other encodings such as HTML and WML. This feature allows peers that do not support XML to access advertised resources.

Advertisements are composed of a series of hierarchically arranged elements. Each element can contain its data or additional elements. An element can also have attributes. Attributes are name-value string pairs. An attribute is used to store meta-data, which helps to describe the data within the element.

Peer Advertisement

A **PeerAdvertisement** describes the peer resources. The primary use of this advertisement is to hold specific information about the peer, such as its name, peer id, registered services and available endpoints.

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PeerAdvertisement>
   <Name> name of the peer</Name>
   <Keywords>search keywords </Keywords>
   <Pid> Peer Id </Pid>
   <Services>
      <jxta:ServiceAdvertisement>
         ...
      </jxta:ServiceAdvertisement>
   </Services>
   <Endpoints>
      <jxta:ServiceAdvertisement>
         ...
      </jxta:EndpointAdvertisement>
```

```
    </Endpoints>
    <InitialApp>
       <jxta:ServiceAdvertisement>
          ...
       </jxta:ServiceAdvertisement>
    </InitialApp>
</jxta:PeerAdvertisement>
```

The peer advertisement contains the following fields:

- Name: This is an optional string that can be associated with a peer. The name is not required to be unique unless the name is obtained from a centralized naming service that guarantees name unicity.
- Keywords: This is an optional string that can be used to index and search for a peer. The string is not guarantee to be unique. Two peers may have the same keywords. The keywords string may contain spaces
- Peer Id: This is a required element that uniquely identifies the peer. Each peer has a unique id. The peer id representation is given in the Id Chapter.
- Service: a service advertisement element for each service published on the peer. Services started on a peer may publish themselves to the peer. Not all services running on the peer need to publish themselves.
- Endpoint: an endpoint URI (tcp://129.144.36.190:9701 or http://129.144.36.190:9702) for each endpoint available on the peer.
- InitialApp: Optional application/service started when the peer is booted. A service advertisement is used to describe the service.

Table 0-1 Peer Advertisement

Element Name	Occurrence	Element Value Type
Name	0/1[1]	String[4]
Keywords	0/1	String
Pid	1	Jxta ID
Services	*[2]	ServiceAdvertisement
Endpoints	+[3]	EndpointAdvertisement
InitialApp	0/1	ServiceAdvertisement

1. 0/1 indicate 0 or 1 element
2. ‘*’ indicate 0 or more elements
3. ‘+’ indicate 1 or more elements
4. The String type in the following is assumed to not contain any XML delimiter characters but may contain blank characters.

PeerGroup Advertisement

A **PeerGroupAdvertisement** describes peergroup specific resources: name, group id, membership, and available peergroup services.

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PeerGroupAdvertisement>
   <Name> name of the peer group</Name>
   <Keywords>search keywords </Keywords>
   <Gid> Peer group Id </Gid>
   <Services>
      <jxta:ServiceAdvertisement>
         ...
      </jxta:ServiceAdvertisement>
   </Services>
   <InitialApp>
      <jxta:ServiceAdvertisement>
         ...
      </jxta:ServiceAdvertisement>
   </InitialApp>
</jxta:PeerGroupAdvertisement>
```

- Name: This is an optional name that can be associated with a peergroup. The name is not required to be unique unless the name is obtained from a centralized naming service that guarantee name unicity.
- Keywords: This is an optional string that can be used to index and search for a peergroup. The string is not guarantee to be unique. Two peergroups may have the same keywords. The keywords string may contain spaces.
- PeerGroup Id: This is a required element that uniquely identifies the peergroup. Each peergroup has a unique id. The peergroup id representation is given in the Id Chapter.

- Service: a service advertisement element for each peergroup service available in the peergroup. Not all peergroup services need to be instantiated when a peer joins a peergroup. It is expected that at least a membership service MUST be available, so the membership service nay implement a null authenticator membership. Everybody can join the group.
- InitialApp: Optional application/service started when a peer is joining a peergroup. A service advertisement is used to describe the service. The InitialAPP MAY be started when the peer is joining a group, it is left at the discretionary of the joining peer to decide to either start or not the peergroup application.

Table 0-2 PeerGroup Advertisement

Element Name	Occurrence	Element Value Type
Name	0/1	String
Keywords	0/1	String
Gid	1	JxtaID
InitialApp	0/1	ServiceAdvertisement
Services	+	ServiceAdvertisement

Pipe Advertisement

A **PipeAdvertisement** describes a pipe communication channel. A pipe advertisement is used by the pipe service to create associated input and output pipe endpoints.

Each pipe advertisement can include an optional symbolic name, to name the pipe and a pipe type to indicate the type of the pipe (point-to-point, propagate, secure, etc.).

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PipeAdvertisement>
   <Name> name of the pipe</Name>
   <Id> Pipe Id </Id>
```

```
   <Type> Pipe Type </Type>
</jxta:PipeAdvertisement>
```

- Name: This is an optional name that can be associated with a pipe. The name is not required to be unique unless the name is obtained from a centralized naming service that guarantee name unicity.
- Pipe Id: This is a required element that uniquely identifies the pipe. Each pipe has a unique id. The pipe id representation is given in the Id Chapter.
- Type: This is an optional pipe type which may be provided to specify the quality of services implemented by the pipe. The following types are defined:
  - RELIABLE (guarantee delivery and ordering, and deliver only once)
  - UNRELIABLE (MAY not arrive at the destination, MAY be delivered more than once to the same destination, MAY arrive in different order)
  - SECURE (reliable and encrypted transfer)

Table 0-3 Pipe Advertisement

Element Name	Occurrence	Element Value Type
Name	0/1	String
Id	1	JxtaID
Type	0/1	UNRELIABLE, RELIABLE, SECURE

Service Advertisement

A **ServiceAdvertisement** describes a JXTA-enabled service. A JXTA-enabled service is a service that uses pipes as primary invocation mechanism. To invoke the service, the peer sends a message to the associated service pipe.

All the built-in core peergroup services (discovery, membership, resolver) are JXTA-enabled services. JXTA-enabled services are published using the above service advertisement.

The service advertisement describes how to invoke and use a service. The pipe advertisement and the access method fields provide the placeholder for any kinds of

service invocation schema that define the valid set of XML messages accepted by the service and the associated message flow. The intent if for the JXTA protocols to be agnostic of service invocation and interoperate with any existing framework.

A service advertisement access method filed MAY refer to a WSDL (www.w3.org/TR/wsdl) , ebXML (www.ebxml.org), UPnP (www.upnp.org) or a client-proxy schema.

For example, a WSDL access method will define messages, which are abstract descriptions of the data being exchanged, and the collections of operations supported by the service using a WSDL schema

A service advertisement MAY contain multiple access method tags as they could be multiple ways to invoke a service. The intent is for the peer to ultimately decide which invocation mechanism to use. Small devices may want to use a small-footprint mechanism or a service framework they already have the code for. Larger devices may decide to download a client-proxy code.

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:ServiceAdvertisement>
    <Name> name of the Service</Name>
    <Version> Version Id </Version>
    <Keywords>search keywords </Keywords>
    <Id> Service Id </Id>
    <Pipe> Pipe advertisement</Pipe>
    <Params> service configuration params </Params>
    <URI> service provider location</URI>
    <Provider> Service Provider</Provider>
    <AccessMethods>
        ...
    </AccessMethods>
</jxta:ServiceAdvertisement>
```

- Name: This is an optional name that can be associated with a service. The name is not required to be unique unless the name is obtained from a centralized naming service that guarantee name unicity.
- Keywords: This is an optional string that can be used to index and search for a service. The string is not guarantee to be unique. Two services may have the same keywords. The keywords string may contain spaces.

- Service Id: This is a required element that uniquely identifies a service. Each service has a unique id. The service id representation is given in the Id Chapter.
- Version: This is a required element that specifies the service version number.
- Provider: This is a required element that gives information about the provider of the service. This will typically be a vendor name.
- Pipe: This is an optional element that specifies a pipe advertisement to be used to create an output pipe to connect to the service. Not all services are REQUIRED to use pipes.
- Params: is a list of configuration parameters available to the peer when invoking the service. The parameter field is optional. Parameters are defined as a list of strings.
- URI: This is an optional parameter that can be used to specify the location of where the code for the service may be found (URI).
- Access Methods. At least one access method is REQUIRED to specify how to invoke the service. Multiple access method tags may be used when multiple access methods are available.

Table 0-4 Service Advertisement

Element Name	**Occurrence**	**Element Value Type**
Name	0/1	String
Keywords	0/1	String
Id	1	JxtaID
Version	1	String
Pipe	0/1	PipeAdvertisement
Params	*	String
URI	0/1	URI
Provider	1	String
AccessMethod	+	AccessMethod

Content Advertisement

A **ContentAdvertisement** describes a content that can be shared in a peer group. A content can be a file, a byte array, code or process state. There is no restrictions on the type of contents that can be represented.

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:ContentAdvertisement>
    <Mimetype> name of the pipe</Mimetype>
    <Size> length</Size>
    <Encoding> type </Encoding>
    <Id> Content Id</Id>
    <RefId> Content Id about </RefId>
    <Document> document </Document>
</jxta:ContentAdvertisement>
```

- id: All contents have an unique id. This is a REQUIRED element.
- size: The total size of the content in bytes. A long (unsigned 64-bits) represented as a string. “-1” indicates that the size is unknown.
- mimetype: The mime type of the content. The type may be unknown.
- encoding: Specifies the encoding used.
- refId - If the advertised content is about another content. The refid is the content ID of the referenced content.

Table 0-5 Content Advertisement

Element Name	Occurrence	Element Value Type
MimeType	1	String
Size	1	String
Encoding	1	String

Table 0-5 Content Advertisement

Element Name	Occurrence	Element Value Type
Id	1	JxtaID
RefId	0/1	JxtaID
Document	**1**	Bytes

EndpointAdvertisement

An **EndpointAdvertisement** describes a peer transport protocol. A peer MAY have many transport protocols. Typically, their will be one peer endpoint for each configured transports (TCP/IP, HTTP, etc). Some peers MAY have multiple network interfaces. Each interface MAY have its own set of endpoints.

The endpoint advertisement is an element in the peer advertisement (see PeerAdvertisement) to describe the endpoints available on the peer.

Endpoints are represented with a virtual endpoint address that embeds all necessary information to create a physical connection to the specific endpoint transport. For example, “tcp://123.124.20.20:1002” or “http://134.125.23.10:6002” are URI strings representing endpoint addresses.

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:EndpointAdvertisement>
   <Name> name of the endpoint</Name>
   <Keywords> search string </Keywords>
   <Address> endpoint logical address </Address>
</jxta:EndpointAdvertisement>
```

The EnpointAdvertisement has the following fields:

- Name: This is an optional name that can be associated with an endpoint. The name is not required to be unique unless the name is obtained from a centralized naming service that guarantee name unicity.
- Keywords: This is an optional string that can be used to index and search for an endpoint. The string is not guarantee to be unique. Two endpoints may have the same keywords.

Table 0-6 EndpointAdvertisement

Element Name	Occurrence	Element Value Type
Name	0/1	String
Keywords	0/1	String
Address	1	URI

JXTA Protocols 5

Peer Discovery Protocol

The Peer Discovery Protocol is used to discover any published peer resources. Resources are represented as advertisements. A resource can be a peer, a peergroup, a pipe, or a service or any resource that has an advertisement. Each resource MUST be represented by an advertisement.

The Peer Discovery Protocol (PDP) enables a peer to find advertisements on other peers. The PDP protocol is the *default* discovery protocol for all user defined peer groups and the world peergroup. Custom discovery services MAY choose to leverage PDP. If a peer group does not have its own discovery service, the PDP will be used as a “lowest common denominator” method for probing peers for advertisements.

The PDP protocol provides at the lowest level, the minimum building blocks for propagating discovery requests between peers. The intent is for PDP to provide the essential discovery infrastructure for building high-level discovery services. In many situation, discovery information is better known by a high-level service, because the service MAY have a better knowledge of the topology (firewall traversal), and the connectivity between peers.

The PDP protocol provides a basic mechanism to discover advertisements while providing hooks so high-level services and applications can participate in the discovery process. Services SHOULD be able to give hints to improve discovery (i.e. decide which advertisements are the most valuable to cache).

The PDP protocol is based on *web-crawling*, and the use of predefined *rendezvous* peers to propagate discovery requests. Rendezvous peers are like any other peers, but they offer to cache advertisements to help others peers discover resources. We can

have as many rendezvous peers as peers in a peergroup. So, not every peers need or can be a rendezvous (no caching capabilities). Only rendezvous peers can forward a discovery request (multi-hop request). A peer will usually be connected to a set of rendezvous peers that help him discover resources.

If a peer does not know the information, it SHOULD ask the surrounding peers (hop of 1) if they know the answer. One peer MAY already have the answer. If no surrounding peers know the answer, the peer MAY ask its rendezvous peers to find advertisements.

The rendezvous peers can forward requests between themselves. The discovery process continues until one rendezvous peer has the answer or the request dies. There is typically a Time To Live (TTL) associated with the request, so it is not infinitely propagated.

For example, suppose a peer A is attempting to discover a resource R on the network. Peer A issues a discovery request specifying the type (peer, peergroup, pipe, service) of advertisements, it is looking for.

To initiate the Discovery, peer A sends a Discovery Request message as a single propagate packet to all its available endpoints. The packet MUST contain the requested peer advertisement, so the receiving peer can respond to the requester. Each Discovery Request identifies the initiator, and an unique request identification specified by the initiator of the request.

When another peer receives the Discovery Request (let's assume peer B in this example), if it has the requested R advertisement, it will return to peer A the advertisement for R in a Discovery Response message.

If Peer A does not get response from its surrounding peers (hop of 1), Peer A MAY send the request to its known rendezvous peers. If the rendezvous peers do not have the advertisement, they can propagate the request to all other rendezvous peers they know. When a rendezvous receives a respond to a request, the rendezvous MAY cache the R advertisement for future usage, before sending it to the requestor.

A peer MUST NOT initiate a new Discovery Request until the minimum allowable interval between Discoveries is reached. This limitation on the maximum rate of Discoveries is similar to the mechanism required by Internet nodes to limit the rate at which ARP requests are sent for any single target IP address.

Messages to Discover Advertisements

Messages to discover advertisements within a peergroup. Message contains peer group credential of probing peer. Identifies probing peer to the message recipient. The destination address is:

- any peer within a region (a propagate message)
- a rendezvous peer (a unicast message)

The response message returns one or more advertisements. There is no guarantee that a response to a discovery query request will be made.

Discovery Query Message

The discovery query message is used to send a discovery request to find advertisements. The discovery query is sent as a query string (tag,value) form. A null query string can be sent to match any results. A threshold value is passed to indicate the maximum number of matches requested by a peer.

```
<?xml version="1.0" encoding="UTF-8"?>
<DiscoveryQueryMsg>
   <Credential> Credential </Credential>
   <QueryId> query id</QueryId>
   <Type> type of request (PEER, GROUP, ADV) </Type>
   <Threshold> requested number of responses </Threshold>
   <PeerAdv> peer advertisement of requestor </PeerAdv>
   <Query> query string (tag, value)</Query>
</DiscoveryQueryMsg>
```

Table 0-7 Discovery Query Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
Type	1	PEER,GROUP,PIPE,SERVICE,ADV
QueryId	1	long
Threshold	1	Int
PeerAdv	1	PeerAdvertisement
Query	1	String

Discovery Response Message

The Discovery response message is used by a peer to respond to a discovery query message.

```
<?xml version="1.0" encoding="UTF-8"?>
<DiscoveryResponseMsg>
   <Credential> Credential </Credential>
   <QueryId> query id</QueryId>
   <Type> type of request (PEER, GROUP, ADV) </Type>
   <Count> number of responses </Count>
   <Adv>
      peer or peer group or pipe or service advertisement
response
   </Adv>
   <............>
   <Adv>
      peer or peer group or pipe or service advertisement
response
   </Adv>
</DiscoveryResponseMsg>
```

Table 0-8 Discovery Response Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
Type	1	PEER,GROUP,PIPE,SERVICE,ADV
QueryId	1	long
Count	1	Int
Adv	+	Advertisement

- Credential: The credential of the sender
- QueryId: Query Id (long as a String)
- Type: specify which advertisements are returned
- count: how many responses

- Adv: advertisement responses.

Peer Resolver Protocol

The Peer Resolver Protocol (PRP) enables a peer to send a generic query to another peer service. Each service can register a handler in the peergroup resolver service to process resolver query requests. Resolver queries are de-multiplexed to each service. Each service can respond to a peer via a resolver response message.

The PRP protocol enables each peer to send and receive generic queries to find or search service information such the state of the service, the state of a pipe endpoint, etc.

Each resolver query has a unique service handler name to specify the receiving service, and a query string to be resolved.

The Resolver protocol provides a generic mechanism for peers to send queries, and receive responses. It removes the burden for registered message handlers by each service, and set message tags to ensure uniqueness of tags. The PRP protocol ensures that messages are sent to correct addresses, and peergroups. It performs authentication, and verification of credentials and drop rogue messages.

There is no guarantee that a response to a resolver query request will be made.

Resolver Query Message

The resolver query message is used to send a resolver query request to another member of a peergroup. The resolver query is sent as a query string to a specific service handler. Each query has a unique Id. The query string can be any string that will be interpreted by the targeted service handler.

```
<?xml version="1.0" encoding="UTF-8"?>
<ResolverQueryMsg>
   <Credential> Credential </Credential>
   <HandlerName> name of handler </HandlerName>
   <QueryId> incremental query Id </QueryId>
   <Query> query string </Query>
</ResolverQueryMsg>
```

The ResolverQueryMsg defines the following fields:

- Credential: The credential of the sender
- QueryId: Query Id (long as a String)
- HandlerName: service the query needs to be passed
- Query: query string

Table 0-9 Resolver Query Message

Element Name	Occurrence	Element Value Type
Credential	Required	Credential
HandlerName	Required	String
QueryId	Required	Int
Query	Required	String

Resolver Response Message

The resolver response message is used to send a resolver response message in response to a resolver query message.

```
<?xml version="1.0" encoding="UTF-8"?>
<ResolverResponseMsg>
   <Credential> Credential </Credential>
   <HandlerName> name of handler </HandlerName>
   <QueryId> query Id </QueryId>
   <Response> response </Response>
</ResolverResponseMsg>
```

The ResolverResponseMsg defines the following fields:

- Credential: The credential of the sender
- QueryId: Query Id (long as a String)
- HandlerName: service the query needs to be passed
- Response: response String

Table 0-10 Resolver Response Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
HandlerName	1	String
QueryId	1	Int
Response	1	String

Peer Information Protocol

Once a peer is located, its capabilities and status may be queries. PIP provides a set of messages to obtain a peer status information.

Messages to Obtain Peer Status

To see if a peer is alive (responding to messages), send it a *ping* message.

- The destination address is the peer's endpoint returned during discovery.
  - Message contains credential of requesting peer. Identifies probing peer to message recipient.
  - Message also contains a ID unique to sender. This ID must be returned in the response.

Messages to Get Peer Info

The peerinfo message gets a list of named properties exported by a peer. All properties are named (by a string), and are "read-only". Higher-level services may offer "read-write" capability to the same information, given proper security credentials.

Each property has a name and a value string. Read-write widgets allow the string value to be changed, while read-only widgets do not. PIP only gives read access.

The destination address is:

- a peer's main endpoint returned in a discovery response message.

Ping Message

The Ping message is sent to a peer to check if the peer is alive and to get information about the peer. The ping option defines the response type returned. A full response (peer advertisement) or a simple acknowledge response (alive and uptime) can be returned.

```
<?xml version="1.0" encoding="UTF-8"?>
<Ping>
   <Credential> Credential </Credential>
   <SourcePid> Source Peer Id </SourcePid>
   <TargetPid> Target Peer Id </TargetPid>
   <Option> type of ping requested</Option>
</Ping>
```

Table 0-11 Ping Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
TargetPid	1	JxtaID
Option	1	String

PeerInfo Message

The peer info response message is used to send a response message in response to a ping message.

```
<?xml version="1.0" encoding="UTF-8"?>
<PeerInfo>
   <Credential> Credential </Credential>
   <SourcePid> Source Peer Id </SourcePid>
   <TargetPid> Target Peer Id </TargetPid>
   <Uptime> uptime</Uptime>
   <TimeStamp> timestamp </TimeStamp>
```

```
   <PeerAdv> Peer Advertisement </PeerAdv>
</PeerInfo>
```

Table 0-12 PeerInfo Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
TargetPid	1	JxtaID
Uptime	1	long
TimeStamp	1	long
PeerAdv	1	PeerAdvertisement

Peer Membership Protocol

The Peer Membership Protocol (PMP) allows a peer to:

- Obtain group membership requirements and application credential.
- Apply for membership and receive a membership credential along with a full group advertisement.
- Update an existing membership or application credential.
- Cancel membership or application credential.

The first step to joining a peergroup is to obtain a credential that is used to become a group member.

Membership Messages

PMP defines the following messages:

- *Apply* - This message is sent by a potential new group member to the group membership application authenticator (its endpoint is listed in the peer group advertisement of every member). A successful response from the group's

authenticator will include an application credential and a group advertisement that lists (at a minimum) the group's membership service. The apply message contains the following fields:

- The current credential of the candidate group member.
- The peer endpoint for the peer group membership authenticator to respond to with an ack message

```
<?xml version="1.0" encoding="UTF-8"?>
<MembershipApply>
    <Credential> Credential of requestor </Credential>
    <SourcePid> Source pipe id</SourcePid>
    <Authenticator> Authenticator pipe adv</Authenticator>
</MembershipApply>
```

Table 0-13 Membership Apply Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
Authenticator	1	PipeAdvertisement

- *Join* - This message is sent by a peer to the peergroup membership authenticator (its endpoint is listed in all peer group advertisements) to join a group. The peer must pass an application credential (from apply response ack msg) for authentication purposes. A successful response from the group's authenticator will include a full membership credential and a full group advertisement that lists (at a minimum) the group's membership configurations requested of full members in good standing. The message contains the following fields:
  - Credential (application credential of the applying peer: See Ack msg) This credential acts as the application form when joining.
  - The peer endpoint for the authenticator to respond to with an Ack message

```
<?xml version="1.0" encoding="UTF-8"?>
<MembershipJoin>
```

```
    <Credential> Credential of requestor </Credential>
    <SourcePid> Source pipe Id </SourcePid>
    <Membersship> membership pipe Advertisement </Membership>
    <Identity> identity</Identity>
</MembershipJoin>
```

Table 0-14 Membership Join Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
Membership	1	PipeAdvertisement
Identity	1	Identity

- *Ack* - This message is an acknowledge message for both join and apply operations. It is sent back by the membership authenticator to indicate whether or nor the peer was granted application rights (peer is applying) or full membership (peer is joining) to the PeerGroup. The message contains the following fields:
  - Credential (application or membership credential allocated to the peer by the peer group authenticator)
  - A more complete peer group advertisement (providing access to further configurations). Not all configuration protocols are visible until the peer has been granted membership or application rights. Some configurations may need to be protected. Also, depending on the peer credential, the peer may not have access to all the configurations.)

```
<?xml version="1.0" encoding="UTF-8"?>
<MembershipAck>
    <Credential> Credential </Credential>
    <SourcePid> Source pipe Id </SourcePid>
    <Membersship> membership pipe adv</Membership>
    <PeerGroupAdv> peer group advertisement </PeerGroupAdv>
    <PeerGroupCredential> credential granted </PeerGroupCredential>
```

```
</MembershipAck>
```

Table 0-15 Membership AckMessage

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
Membership	1	PipeAdvertisement
PeerGroupAdv	1	PeerGroupAdvertisement
PeerGroupCredential	1	Credential

- *Renew* - This message is sent by a peer to renew its credential (membership or application) access to the peer group. An ack message is returned with a new credential and lease (if renew was a "Accepted"). The renew message contains the following fields:
  - Credential (membership or application credential of the peer). The peer endpoint to send an ack response message

```
<?xml version="1.0" encoding="UTF-8"?>
<MembershipRenew>
   <Credential> Credential </Credential>
   <SourcePid> Source pipe Id </SourcePid>
   <Membership> membership pipe Adv</Membership>
</MembershipRenew>
```

Table 0-16 Membership Renew Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
Membership	1	PipeAdvertisement

- *Cancel* - This message is sent by a peer to cancel membership in or application rights in a peer group. The message contains the following fields:
  - Credential (membership or application credential of the peer). The peer endpoint to send an ack message. Successful ack to a cancel contains only a response status of: “Accepted”.

```
<?xml version="1.0" encoding="UTF-8"?>
<MembershipCancel>
   <Credential> Credential </Credential>
   <SourcePid> Source pipe Id </SourcePid>
   <Membersship> membership pipe Adv </Membership>
</MembershipCancel>
```

Table 0-17 Membership Cancel Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
SourcePid	1	JxtaID
Membership	1	PipeAdvertisement

Pipe Binding Protocol

The Pipe Binding Protocol (PBP) is used by peer group members to bind a pipe advertisement to a pipe endpoint. A pipe is conceptually a virtual channel between two pipe endpoints (input and output pipes).

The pipe virtual link (pathway) can be layered upon any number of physical network transport links such as TCP/IP. Each end of the pipe works to maintain the virtual link and to re-establish it, if necessary, by binding or finding the pipe’s currently bound endpoints.

Pipe Implementations and Transport Configurations

A pipe can be *viewed* as an abstract named message queue, supporting create, open/resolve (bind), close (unbind), delete, send, and receive operations. Actual pipe implementations may differ, but all compliant implementations use PBP to bind the pipe to a pipe endpoint. During the abstract create operation, a local peer *binds* a pipe endpoint to a pipe transport.

Each peer that create a pipe, makes an endpoint available (*binds*) to the pipe's transport.

Pipe Messages

PBP defines the following messages:

- *query*: - This message is sent by a peer pipe endpoint to find a pipe endpoint bound to the same pipe advertisement. The message contains the following fields:

```
<?xml version="1.0" encoding="UTF-8"?>
<PipeBindingQuery>
   <Credential> query credential </Credential>
   <Peer>
      optional tag. If present, it includes the URI of the
      only peer that is supposed to answer that request.
   </Peer>
   <Cached> true if the reply can come from a cache </Cached>
   <PipeId> pipe id to be resolved </PipeId>
</PipeBindingQuery>
```

The requestor may ask that the information was not obtained from the cache. This is to obtain the most up-to-date information from a peer to address stale connection.

Table 0-18 Pipe Binding Query Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
Peer	0/1	URI
Cached	1	boolean
PipeId	1	JxtaID

- answer- This response message is sent back to the requesting peer by each peer bound to the pipe. The message contains the following fields:

```
<?xml version="1.0" encoding="UTF-8"?>
<PipeBindingAnswer>
   <Credential> credential </Credential>
   <PipeId> pipe id resolved </PipeId>
   <Peer>
      peer URI where a corresponding InputPipe has been created
   </Peer>
   <Found>
      true: the InputPipe does exist on the specified peer (ACK)
      false: the InputPipe does not exist on the specified peer
             (NACK)
   </Found>
</PipeBindingAnswer>
```

Table 0-19 Pipe Binding Answer Message

Element Name	Occurrence	Element Value Type
Credential	1	Credential
Peer	1	URI
Found	1	boolean
PipeId	1	JxtaID

Endpoint Routing Protocol

The JXTA network is an ad hoc, multi-hops, and adaptive network by nature. Connections in the network may be transient, and message routing is nondeterministic. Routes MAY be unidirectional and change rapidly. Peers MAY appear and leave frequently. A peer behind a firewall can send a message directly to a peer outside a firewall. But a peer outside the firewall cannot establish a connection directly with a peer behind the firewall.

The Endpoint Routing Protocol defines a set of request/query messages, that is processed by a routing service to help a peer route message to its destination.

When a peer is asked to send a message to a given peer endpoint address, it looks in its local cache if it has a route to this peer. If it does not find a route, it sends a route resolver query message to its available *peer routers* asking for a route information. A peer can have as many peer routers as it can find or they can be pre-configured. Any number of peers in a peergroup can elect themselves to become peer routers for other peers. Peers routers offer the ability to cache route information, as well as bridging different physical (different transport) or logical (firewall and NAT) networks. A peer can dynamically found its router peer via a qualified discovery search.

When a peer router receives a route query, if it knows the destination (a route to the destination), it answers the query by returning the route information as an enumeration of hops. The message can be sent to the first router and that router will use the route information to route the message to the destination peer. The route is ordered from the next hop to the final destination peer. At any point the routing information MAY be obsolete requiring the current router to find a new route.

The peer endpoint adds extra routing information in the messages sent by a peer. When a message goes through a peer, the endpoint of that peers leaves its trace onto the message. The trace can be used for loop detection, and to discard recurrent messages. The trace is also used to record new route information by peer routers

ERP provides the last resort routing for a peer. More intelligent routing can be implemented by more sophisticated routing services in place of the core routing service. High-level routing services can manage and optimize routes more efficiently than the core service. JXTA intent is to provide just the hook necessary for user defined routing services to manipulate and update the route table information (route advertisements) used by the peer router. The intent is to have complex route analysis and discovery to be performed above the core by high-level routing services, and have those routine services provide intelligent hints to the peer router to route messages.

The Endpoint Routing Protocol (ERP) is used by a peer router to send messages to another peer router to find the available routes to send a message to a destination peer.

Two communicating peers may need to use a peer router to route messages depending on their network location. For instance, the two peers may be on different transports, or the peers may be separated by a firewall.

Route information are represented as follow:

```
<?xml version="1.0" encoding="UTF-8"?>
<EndpointRouter>
   <Src> peer id of the source </Src>
   <Dest> peer id of the destination </Dest>
   <TTL> time to leave </TTL>
   <Gateway> ordered sequence of gateway </Gateway>
   <...................>
   <Gateway> ordered sequence of gateway </Gateway>
</EndpointRouter>
```

Table 0-20 Endpoint Route

Element Name	Occurrence	Element Value Type
Src	1	JxtaID
Dest	1	JxtaID
TTL	1	int
Gateway	+	URI

The ERP protocol is composed of two messages a route request, and a route answer from the router peer. There is no guarantee that a route response will be received after a query is sent.

Peer routers will typically cache route information. Any peer can query a peer router for route information. Any peer in a peer group MAY become a peer router.

Route Query Request

This message is sent by a peer to a peer router to request a route information. Route information may be cached or not. The query MAY indicate to bypass the cache content of a router, and search dynamically for a new route.

```
<?xml version="1.0" encoding="UTF-8"?>
<EndpointRouterQuery>
   <Credential> credential </Credential>
   <Dest> peer id of the destination </Dest>
   <Cached>
      true: if the reply can be a cached reply
      false: if the reply must not come from a cache
   </Cached>
</EndpointRouterQuery>
```

Table 0-21 Endpoint Router Query

Element Name	Occurrence	Element Value Type
Credential	1	Credential
Dest	1	JxtaID
Cached	1	boolean

Route Answer Request

This message is sent by a router peer to a peer in response to a route information request.

```
<?xml version="1.0" encoding="UTF-8"?>
<EndpointRouterAnswer>
   <Credential> credential </Credential>
   <Dest> peer id of the destination </Dest>
   <RoutingPeer>
```

```
      URI of the router that knows a route to DestPeer
   </RoutingPeer>
   <RoutingPeerAdv>
      Advertisement of the routing peer
   </RoutingPeerAdv>
   <Gateway> ordered sequence of gateway </Gateway>
   < ..................>
   <Gateway> ordered sequence of gateway </Gateway>
</EndpointRouterAnswer>
```

Table 0-22 Endpoint Router Answer

Element Name	Occurrence	Element Value Type
Credential	1	Credential
Dest	1	JxtaID
RoutingPeer	1	URI
RouterPeerAdv	1	PeerAdvertisement
Gateway	+	URI

Messages 6

Introduction

The JXTA protocols assume a low-level XML message transport layer as a basis for providing internet-scalable peer to peer communication. JXTA protocol messages are sent between peer *endpoints*. A peer endpoint is a logical destination (embodied as a URN) on any networking transport capable of sending and receiving datagram-style messages. Endpoints are mapped into physical addresses by the messaging transport layer at runtime.

The intent is for the messaging layer to be compatible with W3C XML messaging standards.

Each peer's endpoint messaging layer delivers an ordered sequence of bytes from one peer to another peer. The messaging layer sends information as a sequence of bytes in one atomic message unit.

The messaging layer uses the transport specified by the URN to send and receive messages. Both reliable connection-based transports such as TCP/IP and unreliable connection less transports like UDP/IP are supported. Other existing message transports such as IRDA, and emerging transports like Bluetooth are easily supported by using the peer endpoint addressing scheme.

Message Format

JXTA endpoint messages are specified as follow:

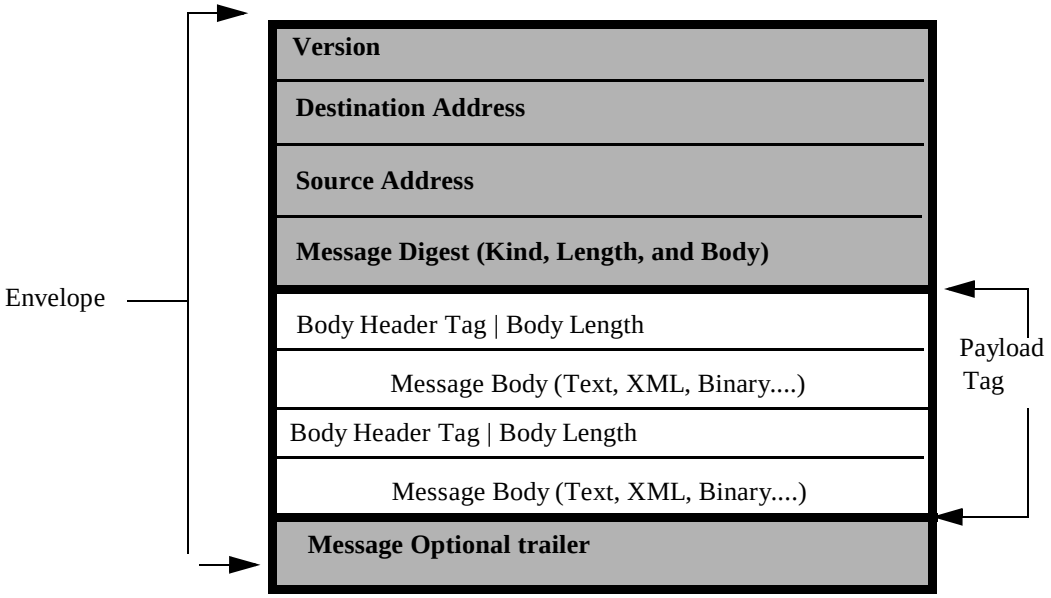
- Contain an envelope, a stack of protocol headers with bodies and an optional trailer

– The envelope contains a header, a message digest, source endpoint, and destination endpoint.

– Each protocol header consists of a tag, naming the protocol in use and a body length.

– Each protocol body is a variable length amount of bytes that is protocol tag dependent.

– the optional trailer may contain traces and accounting information



The following specification of a JXTA peer endpoint message matches the current state of the reference implementation.

It is important to point out that the current message representation is not a well-formed and valid XML document. The document is missing a root element.

```
<JxtaMessageVersion> version number “1.0”</JxtaMessageVersion>
<JxtaMessageDest> destination peer id </JxtaMessageDest>
<JxtaMessageSrc> source peer id </JxtaMessageSrc>
<JxtaMessageDigest> digest </JxtaMessageDigest>
```

```
<JxtaMessageTagName> tag </JxtaMessageTagName>
<JxtaMessageTagData> body </JxtaMessageTagData>
...........
<JxtaMessageTagName> tag </JxtaMessageTagName>
<JxtaMessageTagData> body </JxtaMessageTagData>
<JxtaMessageTrailer> String</JxtaMessageTrailer>
```

The version number is a string “1.0”. The destination and source peer ids are represented as JXTA id’s (See ID chapter for Id representation). The digest is either an MD5 or SHA1 hash. A message can have as many tag parts as needed. The tag name is a String and the body is a byte array containing a string without XML escape characters (“<“, ”>”) or a base64 encoded string.

The message trailer and digest are currently not implemented.

Table 0-23 Endpoint Messages

Element Name	Occurrence	Element Value Type
JxtaMessageVersion	1	String
JxtaMessageDest	1	JXTA ID
JxtaMessageSrc	1	JXTA ID
JxtaMessageDigest	0/1	String[3]
JxtaMessageTagName	+[1]	String
JxtaMessageTagData	*[2]	Bytes
JxtaMessageTrailer	0/1	String

1. ‘+’ indicate 1 or more elements
2. ‘*’ indicate 0 or more elements
3. The String is assumed to not contain any XML delimiter characters (“<“, “>”)

Revision in Progress

The specification of the message format is under review by the JXTA community. There is a consensus that the format must be changed in order to fully support binary data and multi-part message with mime-types.

This message format must:

1. Allow for arbitrary message header fields, including optional header fields.

2. Must allow for data verification of message contents, cryptographic signing of messages.

3. An arbitrary number of named sub-sections which could contain any form of data of any (reasonable) size.

4. Must be “email-safe” such that its contents can be extracted reliably after standard textual transformations committed my E-mail client and server software.

Proposals for this canonical format generally advocate XML. XML accommodates the first 2 requirements inherently. Requirements 3 and 4 must be accommodated by using conventions for encoding data. Four suggestions have been made for requirement 3 :

a. Encode sub-section data using Base64 in the body of an XML tag.

b. Encode sub-section data into XML CDATA sections using extensions to the XML standard to overcome restrictions with CDATA.

c. Encode the entire message as a multipart mime. The XML envelope would be one section and refer to other sections. Message sub-sections could be encoded using Base64.

d. Use BEEP/BXXP : http://www.faqs.org/rfcs/rfc3080.html also has a binding to TCP described in http://www.faqs.org/rfcs/rfc3081.html

All four of these suggestions can satisfy requirement 4.

While an XML based encoding format meets all of the preceding requirements, there are some problems with using it as the exclusive message encoding format. The most commonly identified complaints are:

a. Messages must be parsed in order to determine the length of sub-sections.

b. non-XML data must be encoded using Base64 (or something similar).

c. Overhead concerns for parsing in general.

Suggestions for addressing these concerns have focused around using non-XML message encodings. Suggestions have been:

a. Binary Data Formats : XTalk and IFF

Transport Bindings

TCP/IP Transport

The following section describes the transport binding of the JXTA protocols over TCP/IP. The document describes the message wire format of JXTA endpoint messages over a TCP/IP socket connection.

TCP/IP Wire Format

This section defines the TCP/IP message wire format. Each TCP/IP message is composed of a header and a body.

- Header
- Body

Header

The format of the header is:

Type	Src IP	Src Port	Size	Option	Unused

The header fields are as follow:

Type: 1 byte

- 1 = this is a propagate message.
- 2 = this is a unicast message.

- 3 = This for ACK //unused
- 4 = This is for NACK // unused

Src IP: 4 bytes (IP addresses are in the IPv4 format)

Src Port: 2 bytes (network byte order representation)

Size: 4 bytes body size no counting the header (network byte order representation)

Option: 1 byte option

- HANDCHECK = 1 not implemented yet
- NONBLOCKING = 2 (asynchronous transfer)

Unused: 4 bytes

Body

The format of the body is described in the Message Chapter and represented as a byte array.

Connection States

The TCP/IP binding does not require to maintain any states. The normal operation is for one peer to send a TCP/IP packet to another one, and closes the socket after the packet was send. This is the minimum functionality required to implement unidirectional pipes.

Optional Optimizations

Keep Alive Optimization

If a receiving end decides to keep the connection active (socket "keep alive"), it can return the value 1 (byte) to the sender to tell the sending end that it is keeping the connection alive. The sending end can reuse the same socket to send a new packet.

HTTP Transport

The following section defines the wire message format for the HTTP binding of the JXTA protocols.

An HTTP request format message is composed of a header and a body

```
<HTML>
    <Code> Header </Code>
    <Msg> Body </Msg>
</HTML>
```

Header

The header of the message is defined by the following string:

“1” = OK Request succeeded

“2” = Request Failed

“3” = empty (no body)

“4” = Response. The msg body is not empty

Body

The format of the body is described in the Message Chapter and represented as a string in the HTML request document.

Connection States

- Peer Connection: Before a message can be sent to a HTTP server peer, the HTTP client is required to send a request for connection to the other peer. The request for connection message uses the empty header type. The message is sent using a GET request to the following server URL: http://ip-name:port/reg/client-peerid/. ip-name specifies the IP of the server peer and the port is the corresponding server port number (8080 for example). The server replies with an empty message containing either the ok (1) or failed (2) header type.
- Message Sending: To send a message to another peer server, the client sends a message with the header code (4) and a message body part. The server replies with an ok or failed message. The message is sent to the following URL using the PUT method: http://ip-name:port/snd/. The server replies with an ok or failed response.
- Message Retrieving: To retrieve messages from a peer server, the client sends a GET request message with the empty header tag to the following URL http://ip-name:port/rec/client-peerid/. The server replies with a Failed message or with a Content message containing the messages retrieved.