[MS-P2PPI]:
Peer-to-Peer Presence and Invitation Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
04/08/2008	0.1		Initial Availability.
05/16/2008	0.1.1	Editorial	Revised and edited the technical content.
06/20/2008	0.1.2	Editorial	Revised and edited the technical content.
07/25/2008	0.1.3	Editorial	Revised and edited the technical content.
08/29/2008	0.1.4	Editorial	Revised and edited the technical content.
10/24/2008	0.2	Minor	Updated the technical content.
12/05/2008	0.2.1	Editorial	Revised and edited the technical content.
01/16/2009	0.2.2	Editorial	Revised and edited the technical content.
02/27/2009	0.2.3	Editorial	Revised and edited the technical content.
04/10/2009	0.2.4	Editorial	Revised and edited the technical content.
05/22/2009	0.2.5	Editorial	Revised and edited the technical content.
07/02/2009	0.2.6	Editorial	Revised and edited the technical content.
08/14/2009	0.2.7	Editorial	Revised and edited the technical content.
09/25/2009	0.2.8	Editorial	Revised and edited the technical content.
12/18/2009	0.2.9	Editorial	Revised and edited the technical content.
01/29/2010	0.3	Minor	Updated the technical content.
03/12/2010	0.3.1	Editorial	Revised and edited the technical content.
04/23/2010	0.3.2	Editorial	Revised and edited the technical content.
06/04/2010	1.0	Major	Updated and revised the technical content.
07/16/2010	1.1	Minor	Clarified the meaning of the technical content.
08/27/2010	2.0	Major	Significantly changed the technical content.
10/08/2010	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	2.0	No change	No changes to the meaning, language, or formatting of

Date	Revision History	Revision Class	Comments
			the technical content.

Contents

1 Introduction

This document specifies the Peer-to-Peer Presence and Invitation (P2PPI) Protocol.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

ASCII
binary large object (BLOB)
base64
certificate
certificate authority (CA) or certification authority
Domain Name System (DNS)
globally unique identifier (GUID)
Transmission Control Protocol (TCP)
Transport Layer Security (TLS)

The following terms are specific to this document:

application-defined message: A message that has a format defined by a higher-layer protocol or application.

endpoint: A tuple (composed of an IP address, port, and protocol number) that uniquely identifies a communication **endpoint**.

invitation: A session initiation request.

MIME: A description that identifies the type of a data **binary large object (BLOB)**, as specified in [RFC2046].

name: A string that identifies a **value**.

nickname: A friendly string that identifies the user of the higher-layer application.

node: One of two parties in a P2PPI session.

object: A **name** and **value** pair published by a higher-layer protocol or application.

out of band: An implementation-specific means of obtaining or exchanging data used by the Peer-to-Peer Presence and Invitation Protocol.

peer: A **node** connected to another **node**.

rich presence: A detailed description of the network status (online, offline, busy, away, playing Halo, and so forth) of the user of a system.

session initiation: The process of initiating an application session.

value: A string that contains data supplied by the higher-layer protocol or application.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-PNRP] Microsoft Corporation, "Peer Name Resolution Protocol (PNRP) Version 4.0 Specification", July 2007.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, November 1987, http://www.ietf.org/rfc/rfc1035.txt

[RFC2045] Freed, N., and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996, http://ietf.org/rfc/rfc2045.txt

[RFC2046] Freed, N., and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996, http://ietf.org/rfc/rfc2046.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC2279] Yergeau, F., "UTF-8, A Transformation Format of ISO10646", RFC 2279, January 1998, http://www.ietf.org/rfc/rfc2279.txt

[RFC2459] Housley, R., Ford, W., Polk, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999, http://www.ietf.org/rfc/rfc2459.txt

[RFC3174] Eastlake III, D., and Jones, P., "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001, http://www.ietf.org/rfc/rfc3174.txt

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-PNM] Microsoft Corporation, "People Near Me Protocol", March 2008.

1.3 Overview

P2PPI facilitates **session initiation** between two **peer** computers. The protocol can be used to publish and subscribe to **objects** and to send messages. Objects can be used to describe the **rich presence** of the users of the systems and the capabilities of the computers. Messages can be used to signal the start of an application session.

The two **nodes** in a P2PPI session participate equally. Peers can publish objects, subscribe to objects, and send messages.

An implementation of the P2PPI can accept and maintain simultaneous connections to multiple peers.

The following diagram illustrates a typical P2PPI session between two peers.

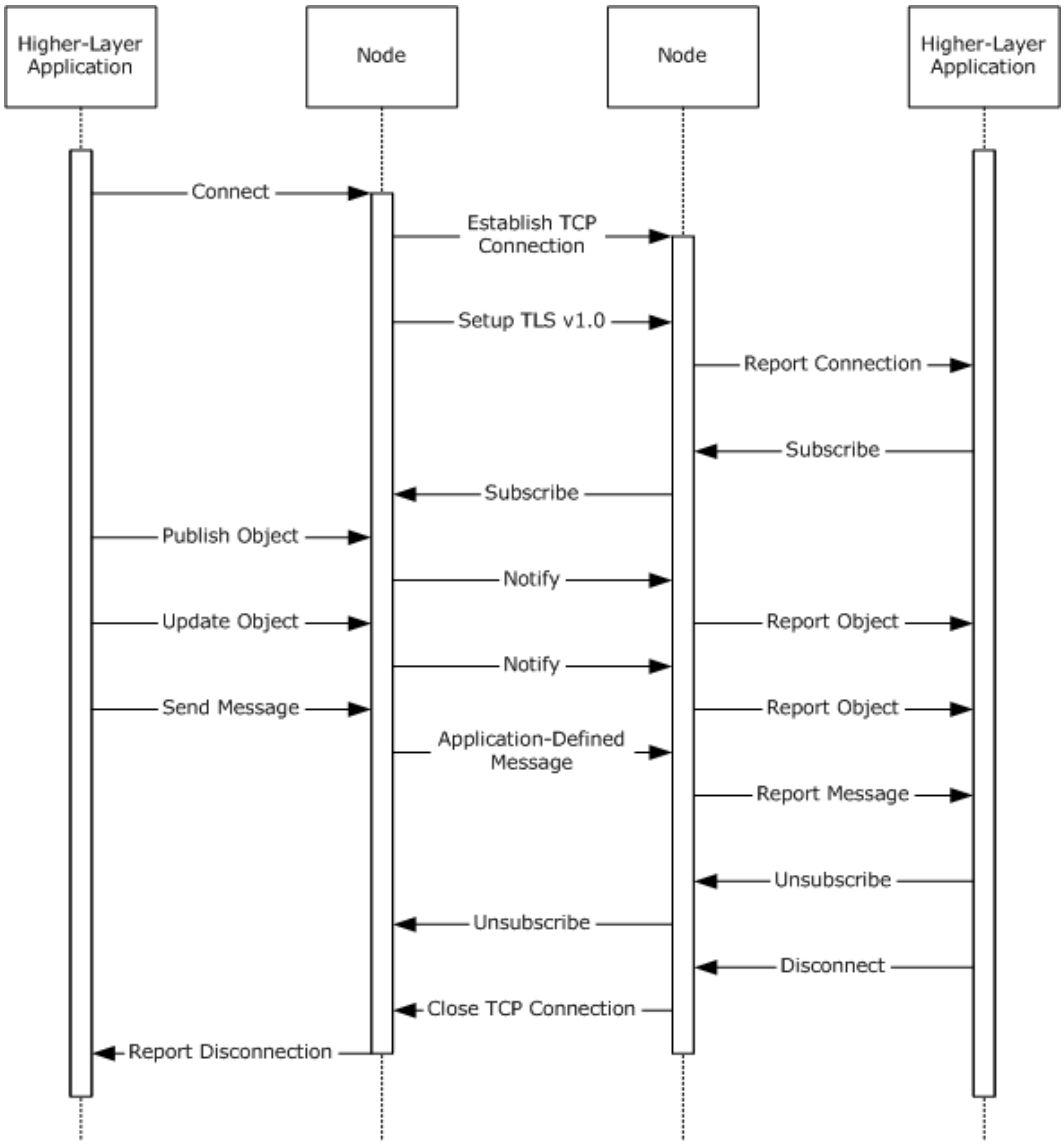


Figure 1: Typical peer-to-peer session

After the initial connection is formed, one node subscribes to a list of objects published by the peer, and the node is notified of the currently published objects. When one of those objects changes, the node is notified about the change. At some point, the node will want to initiate an application session and transmit a message to the peer. Finally, the node will unsubscribe from the objects of the peer and disconnect.

1.4 Relationship to Other Protocols

P2PPI uses **TCP** [RFC793] as a transport. P2PPI does not define a mechanism for authentication or authorization. Communication between peers is encrypted, and mutual authentication is achieved by using the TLS Protocol version 1.0 [RFC2246] as specified in section 2.1.2.

An implementation must use a protocol such as the People Near Me Protocol [MS-PNM], the Peer Name Resolution Protocol [MS-PNRP], or the **Domain Name System (DNS)** [RFC1035] to determine the **endpoint** of peer objects before connecting and using P2PPI between them.

1.5 Prerequisites/Preconditions

1.5.1 Peer Discovery

P2PPI does not define a mechanism for peer discovery. It is assumed that the higher-layer protocol or application determines the endpoint of a peer before invoking P2PPI.

1.6 Applicability Statement

P2PPI is suitable for publishing information needed for session initiation, such as the status of the user of the system and the capabilities of the system. P2PPI is not suitable for use as a transport for bulk data.

1.7 Versioning and Capability Negotiation

P2PPI has no version-negotiation or capability-negotiation behavior, although it carries a protocol version number in its messages.

Protocol Versions: P2PPI messages contain a version number. Only version 1.0 of P2PPI is supported.

1.8 Vendor-Extensible Fields

P2PPI does not make use of any vendor-extensible fields.

1.9 Standards Assignments

None.

2 Messages

P2PPI consists of six types of messages. Each message is composed of one or more fields.

2.1 Transport

2.1.1 Transmission Control Protocol

P2PPI messages MUST be transported over the Transmission Control Protocol (TCP) [RFC793] because P2PPI relies on the message delivery guarantee and delivery order guarantee provided by TCP.

A node MUST use a TCP port numbered 1024 or greater. There is no requirement that a node use a well-known port or that two nodes use the same port number.

2.1.2 Transport Layer Security

P2PPI messages MUST be secured by using the TLS Protocol version 1.0. Mutual authentication and encryption are required.

2.1.2.1 Certificate Format

Clients MUST use X.509 version 3 [RFC2459] format certificates, with constraints on the fields satisfying the description in [MS-PNRP] section 2.2.3.5.

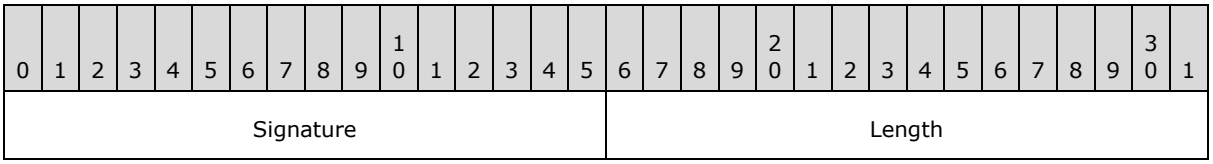
2.2 Message Syntax

All multibyte integer fields in the following sections are defined in network byte order.

2.2.1 Separation Header

P2PPI messages MUST be less than 0xFFF0 bytes in length. P2PPI implements a framing scheme that is independent of the underlying transport.

A P2PPI message consists of a header followed by one or more fields. All P2PPI messages use a common header, as follows.



Signature (2 bytes): This MUST be set to 0x5350.

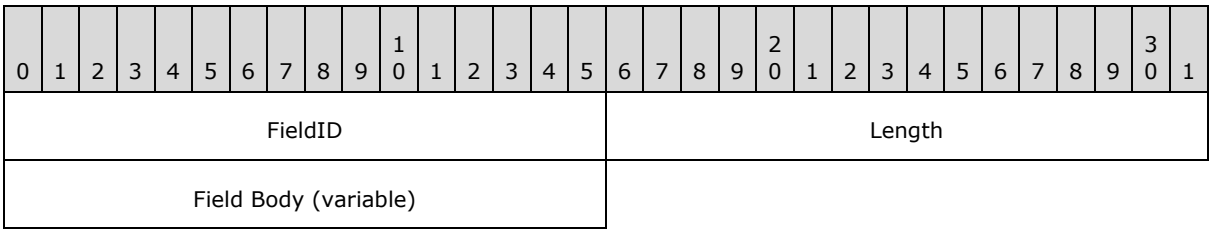
Length (2 bytes): The length, in bytes, of the entire message not including this header.

2.2.2 Fields

P2PPI messages consist of fields. Each field consists of a field header and a field body.

2.2.2.1 Field Header

The field header identifies the field type and field length.



FieldID (2 bytes): The type of field in network byte order. It MUST be one of the values in the following table.

Value	Meaning
MESSAGE_HEADER 0x0100	The field body describes the P2PPI message type. A MESSAGE_HEADER field MUST be the first field in every P2PPI message.
STRING_NAME 0x0201	The field body contains a UTF-8 [RFC2279] string that represents the **name** of an object.
STRING_VALUE 0x0202	The field body contains a UTF-8 string that represents the **value** of an object.
STRING_MIME_TYPE 0x0203	The field body contains a UTF-8 string that represents a **MIME** type.
STRING_MESSAGE 0x0204	The field body contains a UTF-8 string that represents an **application-defined message**.
STRUCTURE_NAME_VALUE 0x0301	The field body contains a STRUCTURE_NAME_VALUE structure.
STRUCTURE_MIME_TYPE_TEXT 0x0302	The field body contains a STRUCTURE_MIME_TYPE_TEXT structure.
ARRAY_NAME_VALUE_LIST 0x0401	The field body contains zero or more STRUCTURE_NAME_VALUE structures.
ARRAY_NAME_LIST 0x0402	The field body contains zero or more STRING_NAME fields.

Length (2 bytes): The length of the field, in bytes, including the field header.

Field Body (variable): Holds field data.

2.2.2.2 Field Body

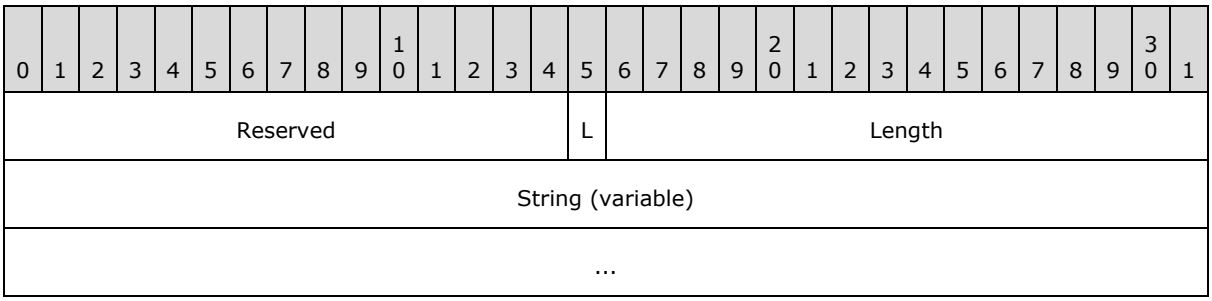
The field body consists of four fields:

- STRING_NAME
- STRING_VALUE
- STRING_MIME_TYPE
- STRING_MESSAGE

Each of these is described in the following sections.

2.2.2.2.1 STRING_NAME Field

The STRING_NAME field identifies the messages.



Reserved (15 bits): This MUST be zeroed when sent and ignored on receipt.

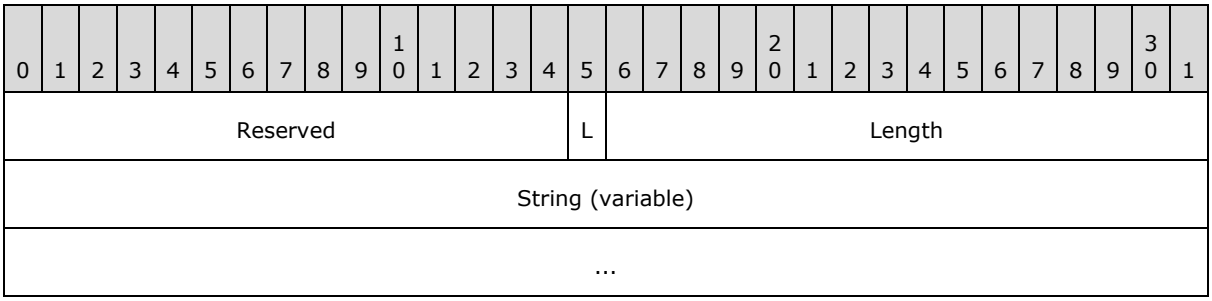
L (1 bit): If clear, this indicates that Length is 0x0000. If set, this indicates that Length is greater than 0x0000.

Length (2 bytes): The length, in bytes, of the String field.

String (variable): A non-NULL-terminated UTF-8 string of zero or more characters describing a name.

2.2.2.2.2 STRING_VALUE Field

The STRING_VALUE field.



Reserved (15 bits): This MUST be zeroed when sent and ignored on receipt.

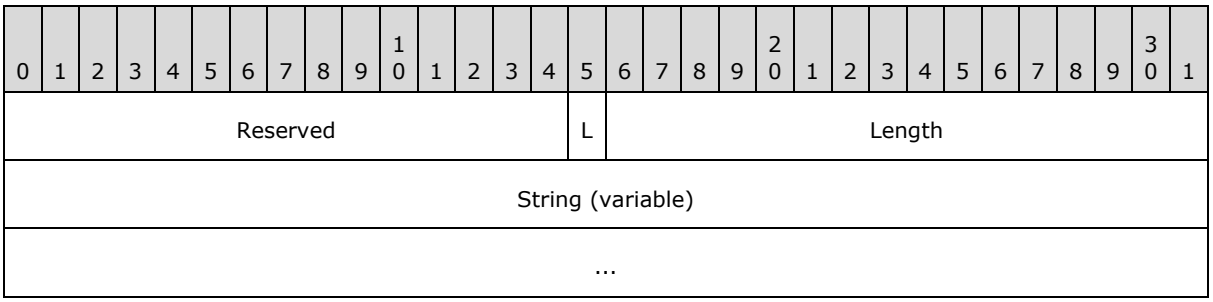
L (1 bit): If clear, this indicates that Length is 0x0000. If set, this indicates that Length is greater than 0x0000.

Length (2 bytes): The length, in bytes, of the String field.

String (variable): A non-NULL-terminated UTF-8 string of zero or more characters describing a name.

2.2.2.2.3 STRING_MIME_TYPE Field

The STRING_MIME_TYPE field identifies the message MIME type.



Reserved (15 bits): This MUST be zeroed when sent and ignored on receipt.

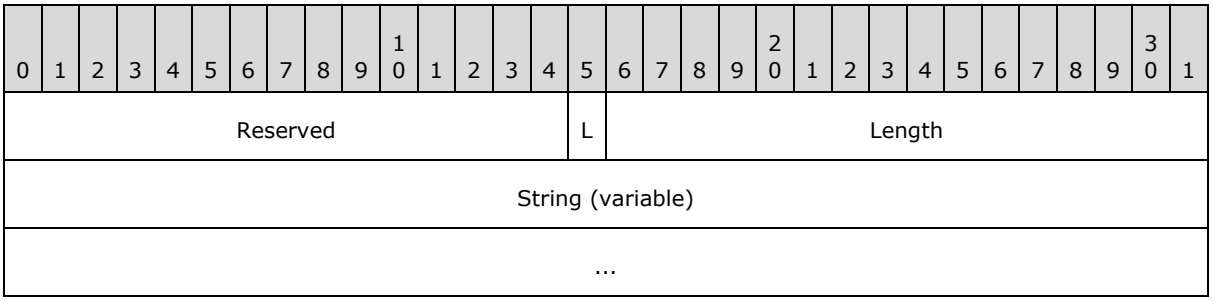
L (1 bit): If clear, this indicates that Length is 0x0000. If set, this indicates that Length is greater than 0x0000.

Length (2 bytes): The length, in bytes, of the String field.

String (variable): A non-NULL-terminated UTF-8 string of zero or more characters describing a MIME type.

2.2.2.2.4 STRING_MESSAGE Field

The STRING_MESSAGE field.



Reserved (15 bits): This MUST be zeroed when sent and ignored on receipt.

L (1 bit): If clear, this indicates that Length is 0x0000. If set, this indicates that Length is greater than 0x0000.

Length (2 bytes): The length, in bytes, of the String field.

String (variable): A non-NULL-terminated UTF-8 string of zero or more characters containing an APPLICATION_DEFINED_MESSAGE message.

2.2.2.3 Structure Field Bodies

A structure field consists of several subfields. P2PPI defines two structures.

2.2.2.3.1 STRUCTURE_NAME_VALUE Field Body

A STRUCTURE_NAME_VALUE field body is used to represent an object published by a node.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
STRING_NAME (variable)																															
...																															
STRING_VALUE (variable)																															
...																															

STRING_NAME (variable): A STRING_NAME field, defining the name of the object.

STRING_VALUE (variable): A STRING_VALUE field, holding the value of the object.

2.2.2.3.2 STRUCTURE_MIME_TYPE_TEXT Field Body

A STRUCTURE_MIME_TYPE_TEXT field body makes up the body of an APPLICATION_DEFINED_MESSAGE message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
STRING_MIME_TYPE (variable)																															
...																															
STRING_VALUE (variable)																															
...																															

STRING_MIME_TYPE (variable): A STRING_MIME_TYPE field, defining the MIME type of the data in the STRING_VALUE field that follows.

STRING_VALUE (variable): A STRING_VALUE field, holding application-supplied data.

2.2.2.4 Array Field Bodies

An array consists or several fields of the same type. P2PPI defines two types of arrays.

2.2.2.4.1 ARRAY_NAME_VALUE_LIST Field Body

An ARRAY_NAME_VALUE_LIST field body is used to transmit a list of objects.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Entries																STRUCTURE_NAME_VALUE List (variable)															

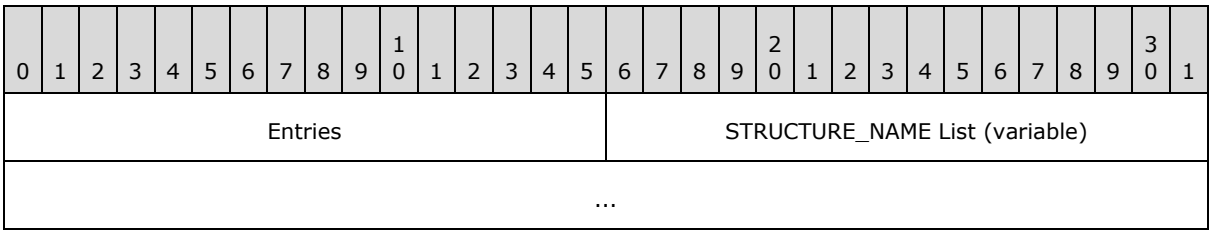
...

Entries (2 bytes): The (unsigned) number of elements in the array.

STRUCTURE_NAME_VALUE List (variable): Zero or more STRUCTURE_NAME_VALUE fields concatenated.

2.2.2.4.2 ARRAY_NAME_LIST Field Body

An ARRAY_NAME_LIST field body is used to transmit a list of object names.

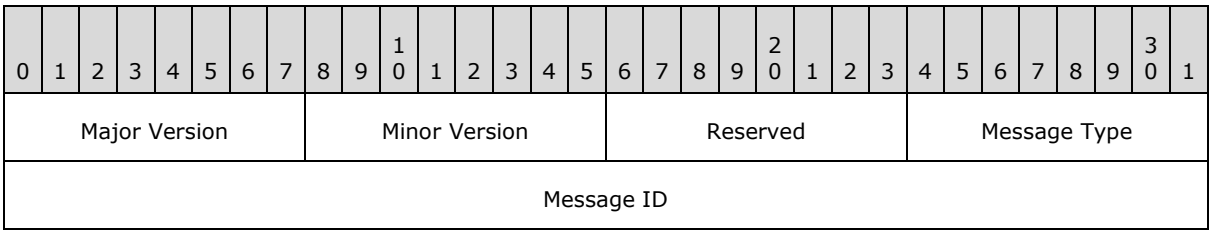


Entries (2 bytes): The (unsigned) number of elements in the array.

STRUCTURE_NAME List (variable): Zero or more STRING_NAME fields concatenated.

2.2.2.5 MESSAGE_HEADER Field Body

Each P2PPI message MUST begin with a MESSAGE_HEADER field. The MESSAGE_HEADER field identifies the protocol version and type of the message.



Major Version (1 byte): This MUST be set to 0x01.

Minor Version (1 byte): This MUST be set to 0x00.

Reserved (1 byte): This MUST be set to 0x00 and ignored upon receipt.

Message Type (1 byte): This identifies the type of the message that follows and MUST be set to one of the values in the following table.

Value	Meaning
APPLICATION_DEFINED_MESSAGE 0x01	The message is an APPLICATION_DEFINED_MESSAGE message.
NOTIFY_MESSAGE 0x02	The message is a NOTIFY_MESSAGE message.
SUBSCRIBE_MESSAGE	The message is a SUBSCRIBE_MESSAGE message.

Value	Meaning
0x03	
UNSUBSCRIBE_MESSAGE 0x04	The message is an UNSUBSCRIBE_MESSAGE message.
REQUEST_MESSAGE 0x05	The message is a REQUEST_MESSAGE message.
RESPONSE_MESSAGE 0x06	The message is a RESPONSE_MESSAGE message.

Message ID (4 bytes): A counter that can be used to keep track of message order.

2.2.3 Objects

P2PPI includes a definition for several types of objects that can be carried in STRUCTURE_NAME_VALUE structures (section 2.2.2.3.1).

This section defines the objects that are used by P2PPI and specifies the format of the **STRING_NAME** and **STRING_VALUE** fields for each object.

2.2.3.1 RICH_PRESENCE Object

The RICH_PRESENCE object communicates the rich presence of the user of the higher-layer application or protocol.

STRING_NAME field: The *String* element MUST be set to "1d6ccc02-3ec4-453b-b986-470b610cb958".

STRING_VALUE field: The *String* element carries an arbitrary string describing the rich presence of the user of the system.

2.2.3.2 CAPABILITY Object

Capability objects identify applications that are installed on the machine. A node can request that a peer start one of these applications by using an **invitation**.

STRING_NAME field: The *String* element MUST be set to "422d4780-5b0e-4355-b1f6-388abdd8d74b".

STRING_VALUE field: The *String* element MUST be set to a **globally unique identifier (GUID)** encoded as a UTF-8 string representing the application as specified in [MS-DTYP] section 2.3.2.3.

2.2.3.3 APPLICATION_DEFINED_OBJECT Object

An arbitrary object published by the higher-layer application.

STRING_NAME field: The *String* element MUST be set to "94e2f051-5d71-43d2-9b7e-e3f8c48f3bab".

STRING_VALUE field: The *String* element carries a UTF-8 string having a format understood by the higher-layer application.

2.2.3.4 CONTACT_DATA Object

Contact data objects hold data describing the user of the system.

STRING_NAME field: The <String> element MUST be set to "ec0b3811-f3eb-4fca-b7f3-19f871aa7d27".

STRING_VALUE field: The <String> element contains information about the user of the system that publishes this object. It MUST conform to the following XML schema.

2.2.3.4.1 CONTACT_DATA Schema

The following XML schema describes the user of the system that publishes the CONTACT_DATA Object.

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="CONTACTINFO" type="ContactInfo"/>
   <xs:complexType name="ContactInfo">
     <xs:sequence>
       <xs:element name="PeerName" type="xs:string"/>
       <xs:element name="NickName" type="xs:string"/>
       <xs:element name="DisplayName" type="xs:string"/>
       <xs:element name="EmailAddress" type="xs:string"/>
       <xs:element name="Credentials" type="xs:binary"/>
     </xs:sequence>
   </xs:complexType>
</xs:schema>
```

PeerName element: This element MUST be a 160-bit SHA-1 [RFC3174] hash of the public key associated with the identity of this peer encoded as a 40-byte **ASCII** string containing the hexadecimal digits representing this hash.

NickName element: A **nickname** that identifies the publisher of the object.

DisplayName element: The name of the user of the higher-layer application publishing this object.

EmailAddress element: A string that represents the e-mail address of the user of the higher-layer application or protocol publishing this object.

Credentials element: A **base64**-encoded X.509 version 3 [RFC2459] format certificate identifying the user, with constraints on the fields satisfying the description in [MS-PNRP] section 2.2.3.5.

2.2.3.5 ENDPOINT_ID Object

The ENDPOINT_ID object contains a GUID that can be used by the higher-layer application to identify the P2PPI session.

STRING_NAME field: The <String> element MUST be set to "2c5ebc42-2557-4217-85a5-eced263d471d".

STRING_VALUE field: The <String> element MUST contain a GUID encoded as ASCII text.

2.2.3.6 USER_PICTURE Object

A user picture object contains a bitmap image representing the user of the system.

STRING_NAME field: The *String* element MUST be set to "dd15f41f-fc4e-4922-b035-4c06a754d01d".

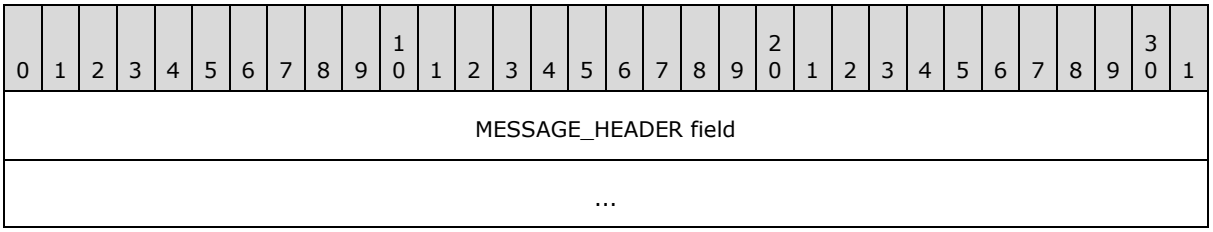
STRING_VALUE field: The *String* element MUST contain base64-encoded [RFC2045] bitmap image data.

2.2.4 Messages

P2PPI messages consist of one or more fields. Every P2PPI message MUST begin with a separation header, which MUST be followed by a **MESSAGE_HEADER** field. A message body follows, which consists of zero or more fields.

2.2.4.1 SUBSCRIBE_MESSAGE Message

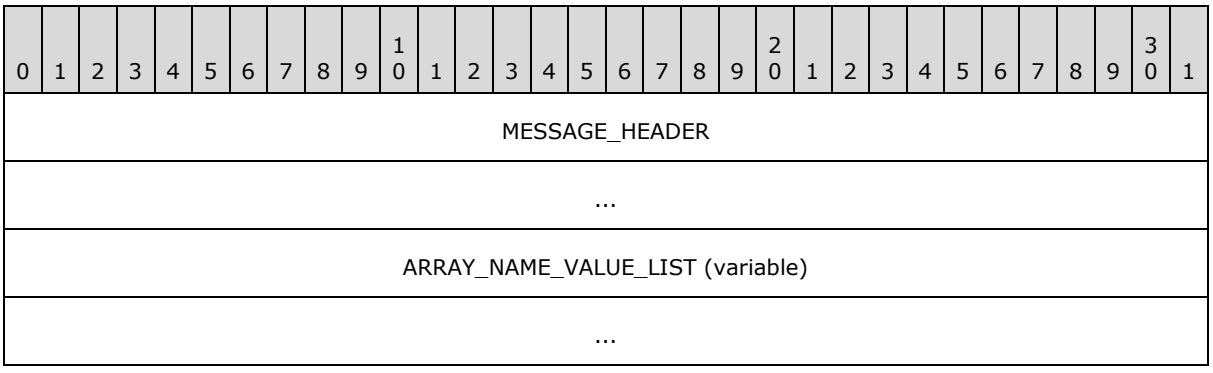
A SUBSCRIBE_MESSAGE message is sent to retrieve objects published by a peer and to subscribe to notification of future changes to those objects and notification of the creation and deletion of objects. A SUBSCRIBE_MESSAGE message consists only of a message header.



MESSAGE_HEADER field (8 bytes): A MESSAGE_HEADER field body (section 2.2.2.5). The *Message Type* element MUST be set to SUBSCRIBE_MESSAGE.

2.2.4.2 NOTIFY_MESSAGE Message

A NOTIFY_MESSAGE message is sent to retrieve the latest list of objects published by a peer. A REQUEST_MESSAGE message does not subscribe to further changes to objects or the creation and deletion of objects, unlike the SUBSCRIBE_MESSAGE Message. A REQUEST_MESSAGE message consists only of a message header.

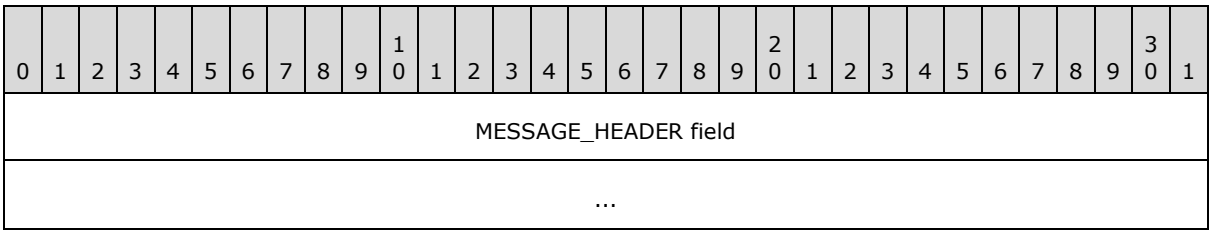


MESSAGE_HEADER (8 bytes): A MESSAGE_HEADER field body (section 2.2.2.5). The <Message Type> element MUST be set to NOTIFY_MESSAGE.

ARRAY_NAME_VALUE_LIST (variable): An **ARRAY_NAME_VALUE_LIST** field consisting of zero or more **STRUCTURE_NAME_VALUE** fields, each defining an object.

2.2.4.3 UNSUBSCRIBE_MESSAGE Message

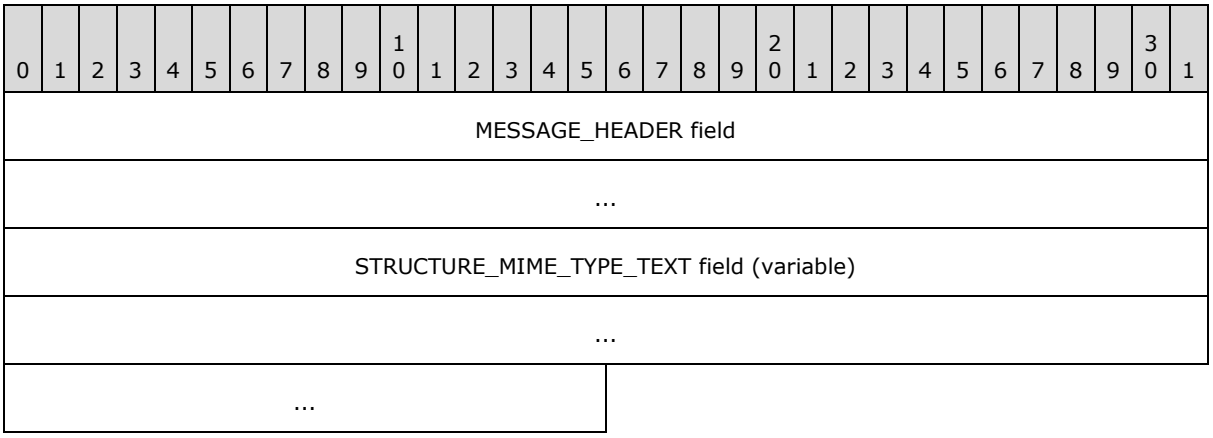
An UNSUBSCRIBE_MESSAGE message is used to unsubscribe from notification of changes to objects and the creation and deletion of objects. An UNSUBSCRIBE_MESSAGE message consists only of a message header.



MESSAGE_HEADER field (8 bytes): A MESSAGE_HEADER field body (section 2.2.2.5). The *Message Type* element MUST be set to UNSUBSCRIBE_MESSAGE.

2.2.4.4 APPLICATION_DEFINED_MESSAGE Message

An APPLICATION_DEFINED_MESSAGE message is sent to retrieve the latest list of objects published by a peer. A REQUEST_MESSAGE message does not subscribe to further changes to objects or the creation and deletion of objects, unlike the SUBSCRIBE_MESSAGE message. A REQUEST_MESSAGE message consists only of a message header.

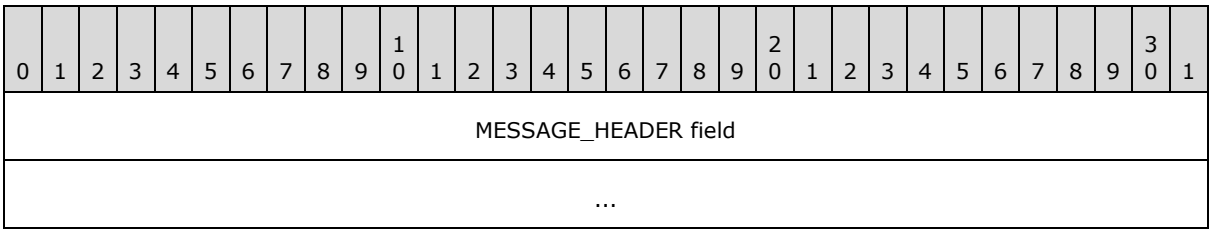


MESSAGE_HEADER field (8 bytes): A MESSAGE_HEADER field body (section 2.2.2.5). The *Message Type* element MUST be set to APPLICATION_DEFINED_MESSAGE.

STRUCTURE_MIME_TYPE_TEXT field (variable): A STRUCTURE_MIME_TYPE_TEXT field, containing the data that makes up the message.

2.2.4.5 REQUEST_MESSAGE Message

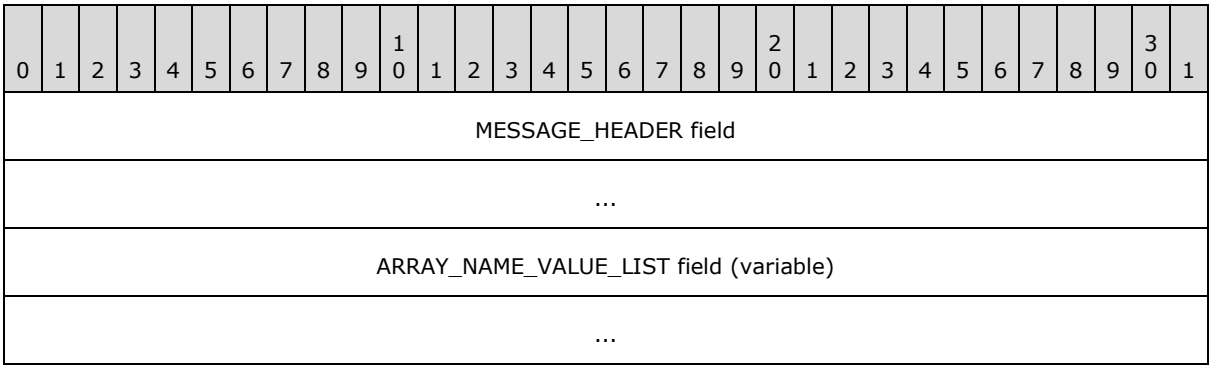
A REQUEST_MESSAGE message is sent to retrieve the latest list of objects published by a peer. A REQUEST_MESSAGE message does not subscribe to further changes to objects or the creation and deletion of objects, unlike the SUBSCRIBE_MESSAGE message. A REQUEST_MESSAGE message consists only of a message header.



MESSAGE_HEADER field (8 bytes): A MESSAGE_HEADER field body (section 2.2.2.5). The *Message Type* element MUST be set to REQUEST_MESSAGE.

2.2.4.6 RESPONSE_MESSAGE Message

A RESPONSE_MESSAGE message is used to transmit a set of published objects. It is sent in response to a REQUEST_MESSAGE message and is functionally equivalent to a NOTIFY_MESSAGE message.



MESSAGE_HEADER field (8 bytes): The standard MESSAGE_HEADER field that MUST be present in all messages. The *Message Type* element MUST be set to RESPONSE_MESSAGE (0x06).

ARRAY_NAME_VALUE_LIST field (variable): An ARRAY_NAME_VALUE_LIST field consisting of zero or more STRUCTURE_NAME_VALUE fields, each defining an object.

2.2.5 Invitations

A peer can send an invitation to start an application by sending an INVITATION_REQUEST message. A peer can send a response to an INVITATION_REQUEST message by sending an INVITATION_ACKNOWLEDGEMENT message. INVITATION_REQUEST and INVITATION_ACKNOWLEDGEMENT are delivered as payloads in APPLICATION_DEFINED_MESSAGE messages. The protocol does not provide any mechanisms for verifying INVITATION_REQUEST messages or INVITATION_ACKNOWLEDGEMENT messages. These messages are passed to the higher-layer protocol or application in the same manner as all APPLICATION_DEFINED_MESSAGE messages.

Each APPLICATION_DEFINED_MESSAGE message contains a STRUCTURE_MIME_TYPE_TEXT field body. This section specifies the format of the STRING_MIME_TYPE and STRING_VALUE fields that make up the STRUCTURE_MIME_TYPE_TEXT field body for INVITATION_REQUEST and INVITATION_RESPONSE.

2.2.5.1 INVITATION_REQUEST

STRING_MIME_TYPE field: The <String> element MUST be set to "text/appinvite".

STRING_VALUE field: The <String> element contains the data that makes up the invitation. It MUST conform to the following XML schema.

2.2.5.1.1 INVITATION_REQUEST SCHEMA

The following XML schema is used to describe the invitation.

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:simpleType name="GUID">
    <xs:annotation>
      <xs:documentation xml:lang="en">
       The representation of a GUID, generally the id of an element.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-
[a-fA-F0-9]{12}"/>
    </xs:restriction>
   </xs:simpleType>

   <xs:element name="PEERINVITE" type="PeerInvite"/>
    <xs:complexType name="PeerInvite">
     <xs:sequence>
       <xs:element name="INVITATIONID" type="xs:GUID"/>
       <xs:element name="APPID" type="xs:GUID"/>
       <xs:element name="MESSAGE" type="xs:string"/>
       <xs:element name="SENDERNICKNAME" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:schema>
```

INVITATIONID element: A GUID that uniquely identifies the invitation. This GUID is used to correlate an INVITATION_RESPONSE message with an INVITATION_REQUEST message.

APPID element: A GUID that uniquely identifies the application to be started in response to this invitation. This GUID is determined by the author of the application to be started.

MESSAGE element: A string that relays an application-defined message between the users of the two systems.

SENDERNICKNAME element: A nickname that identifies the sender of the invitation.

2.2.5.2 INVITATION_ACKNOWLEDGEMENT

STRING_MIME_TYPE field: The *String* element MUST be set to "text/appinvite".

STRING_VALUE field: The *String* element contains the data that makes up the invitation acknowledgment. It MUST conform to the following XML schema.

2.2.5.2.1 INVITATION_ACKNOWLEDGEMENT SCHEMA

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:simpleType name="GUID">
    <xs:annotation>
      <xs:documentation xml:lang="en">
       The representation of a GUID, generally the id of an element.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-
[a-fA-F0-9]{12}"/>
    </xs:restriction>
   </xs:simpleType>

   <xs:element name="PEERINVITE" type="PeerInvite"/>
    <xs:complexType name="PeerInvite">
     <xs:sequence>
       <xs:element name="INVITATIONID" type="xs:GUID"/>
       <xs:element name="RESPONSE" type="xs:integer"/>
       <xs:element name="EXTENDEDINFO" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:schema>
```

<INVITATIONID> element: A GUID that uniquely identifies the invitation. This GUID is used to correlate an INVITATION_RESPONSE message with an INVITATION_REQUEST message.

<RESPONSE> element: An integer that indicates the response to the invitation. The value 1 indicates that the invitation has been accepted. The value 2 indicates that the invitation has been refused. Other integer values are not valid.

<EXTENDEDINFO> element: A string of no more than 255 characters that contains additional data supplied by the higher-layer protocol or application about the application session.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Neighbor subscription flag: The subscription status of the connected peer. The flag indicates whether the peer is in the "subscribed" state or the "unsubscribed" state.

Local subscription flag: The node's subscription status to the connected peer. The flag indicates whether the node is in the "subscribed" state or the "unsubscribed" state.

Response conversation flag: The flag indicates whether the node has sent a REQUEST_MESSAGE message and expects to receive a RESPONSE_MESSAGE message.

Published objects list: A list of objects published by the higher-layer protocol or application. Each object has two components: a string name and a string value. For publication, an object must be translated into a STRUCTURE_NAME_VALUE field body (section 2.2.2.3.1), using the name of the object to complete the embedded STRING_NAME field and the value of the object to complete the embedded STRING_VALUE field. To transmit a published object to a peer, a node embeds that object in the ARRAY_NAME_VALUE_LIST field body (section 2.2.2.4.1) payload of a NOTIFY_MESSAGE message or a RESPONSE_MESSAGE message. The higher-layer application or protocol can publish any of the objects in section 2.2.4 or can define and publish new objects whose names are not listed in section 2.2.4.

3.1.2 Timers

None.

3.1.3 Initialization

The neighbor subscription flag and local subscription flag MUST initially be set to unsubscribed, and the published objects list MUST be empty.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Peer Connection

The higher-layer protocol or application can request a connection to another node. The higher-layer protocol or application MUST supply the endpoint of the peer. The node MUST establish a TCP connection and complete the **TLS** 1.0 handshake as described in section 2.1.2.

P2PPI messages cannot be transmitted until the underlying TCP connection has been established and the TLS setup is finished. The higher-layer protocol or application SHOULD be signaled when the connection setup is finished.

3.1.4.2 Object Publication

The higher-layer protocol or application can publish a new object, delete a published object, or update a published object at any time after the peer connection has been established.

3.1.4.2.1 Publishing a New Object

When the higher-layer protocol or application publishes a new object, the protocol MUST add the object to the published objects list if that object is not already present in the published objects list. If the neighbor subscription flag is set, the protocol MUST send the peer a NOTIFY_MESSAGE message (section 2.2.4.2) that contains the new object. If the neighbor subscription flag is not set, the protocol MUST NOT send a NOTIFY_MESSAGE message to the peer.

3.1.4.2.2 Deleting an Object

When the higher-layer protocol or application deletes an object, the protocol MUST check whether the object was present in the published objects list. If the object was not present, the protocol MUST report an error to the higher-layer application and MUST NOT send a NOTIFY_MESSAGE message to the peer. If the object was present, the protocol MUST remove the object from the published objects list. If the neighbor subscription flag is set, the protocol MUST send the peer a NOTIFY_MESSAGE message (section 2.2.4.2) that contains the entire published objects list. If the neighbor subscription flag is not set, the protocol MUST NOT send a NOTIFY_MESSAGE message to the peer.

3.1.4.2.3 Updating an Object

If the higher-layer protocol or application updates an object, the protocol MUST check whether the object was present in the published objects list. If the object was not present, the protocol MUST report an error to the higher-layer application and MUST NOT send a NOTIFY_MESSAGE message to the peer. If the object was present, the protocol MUST update the object in the published objects list, and if the neighbor subscription flag is set, the protocol MUST send the peer a NOTIFY_MESSAGE message (section 2.2.4.2) that contains the entire published objects list. If the neighbor subscription flag is not set, the protocol SHOULD NOT send a NOTIFY_MESSAGE message to the peer.

3.1.4.3 Subscription

The higher-layer protocol or application can request subscription to the connected peer at any time at any time after the peer connection has been established. When an application requests subscription, the protocol MUST check the local subscription flag. If the local subscription flag has not been set, the protocol MUST set the flag and send a SUBSCRIBE_MESSAGE message (section 2.2.4.1) to the peer. If the flag has already been set, the node MUST set the response conversation flag and send a REQUEST_MESSAGE message instead.

3.1.4.4 Unsubscription

The higher-layer application can unsubscribe at any time after the peer connection has been established and subscription has been previously requested. If the higher-layer protocol or application requests unsubscription and if the local subscription flag has not been set, the protocol SHOULD report an error to the higher-layer application and MUST NOT send an UNSUBSCRIBE_MESSAGE message to the peer. If the local subscription flag has been set, the protocol MUST unset the local subscription flag and send an UNSUBSCRIBE_MESSAGE message (section 2.2.4.3).

3.1.4.5 Object Retrieval

The higher-layer protocol or application can retrieve the list of objects published by the peer without subscribing to that peer. If the higher-layer protocol or application requests object list retrieval, the protocol MUST set the response conversation flag and send a REQUEST_MESSAGE message.

3.1.4.6 Application-Defined Message

The higher-layer protocol or application can request the transmission of an APPLICATION_DEFINED_MESSAGE message at any time after the peer connection has been established. When the higher-layer protocol or application supplies the message contents and the MIME type of the message contents, the protocol MUST compose an APPLICATION_DEFINED_MESSAGE message (section 2.2.4.4). It MUST use the supplied MIME type and the supplied message contents to complete the STRING_MIME_TYPE field and the STRING_VALUE field, respectively, of the STRUCTURE_MIME_TYPE_TEXT field body carried in the APPLICATION_DEFINED_MESSAGE message.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Receiving Peer-to-Peer Presence and Invitation Protocol Messages

When a node receives a P2PPI message, it MUST first check whether the message starts with a separation header that conforms to the syntax as specified in section 2.2.1. If the message does not start with such a separation header, the node MUST drop the message, close the TCP connection, and report an error to the higher-layer protocol or application.

The node MUST then verify that the message begins with a **MESSAGE_HEADER** field that conforms to the syntax specified in section 2.2.2.5. If the message does not begin with such a **MESSAGE_HEADER** field, the node MUST drop the message, close the TCP connection, and report an error to the higher-layer protocol or application.

The node MUST then check the <Message Type> element of the **MESSAGE_HEADER** field and handle additional type-specific processing required by the message type. Messages with types other than those specified in this document MUST be silently dropped.

Message	Acknowledgment message
SUBSCRIBE_MESSAGE	NOTIFY_MESSAGE
APPLICATION_DEFINED_MESSAGE	None
UNSUBSCRIBE_MESSAGE	None
NOTIFY_MESSAGE	None
REQUEST_MESSAGE	RESPONSE_MESSAGE
RESPONSE_MESSAGE	None

3.1.5.2 Receiving a SUBSCRIBE_MESSAGE Message

When a node receives a SUBSCRIBE_MESSAGE message (section 2.2.4.1), it MUST perform the following steps:

1. Verify that the SUBSCRIBE_MESSAGE message correctly matches the format specified in section 2.2.4.1, and silently drop the message if it does not correctly match the format.

2. If the neighbor subscription flag has already been set, the node MUST silently drop the message.

3. If the neighbor subscription flag has not been set, the node MUST set the neighbor subscription flag and prepare a NOTIFY_MESSAGE message that contains the complete published objects list, even if the published objects list is empty. The node MUST then transmit this NOTIFY_MESSAGE message to the peer.

3.1.5.3 Receiving a REQUEST_MESSAGE Message

When a node receives a REQUEST_MESSAGE message, it MUST perform the following steps:

1. Verify that the REQUEST_MESSAGE message correctly matches the format specified in section 2.2.4.5, and silently drop the message if it does not correctly match the format.

2. Prepare a RESPONSE_MESSAGE message that contains the complete published objects list, even if the published objects list is empty. The node MUST then transmit this RESPONSE_MESSAGE message to the peer.

3.1.5.4 Receiving an UNSUBSCRIBE_MESSAGE Message

When a node receives an UNSUBSCRIBE_MESSAGE message, it MUST perform the following steps:

1. Verify that the UNSUBSCRIBE_MESSAGE message correctly matches the format specified in section 2.2.4.3, and silently drop the message if it does not correctly match the format.

2. Clear the neighbor subscription flag.

3.1.5.5 Receiving a NOTIFY_MESSAGE Message

When a node receives a NOTIFY_MESSAGE message, it MUST perform the following steps:

1. Verify that the NOTIFY_MESSAGE message correctly matches the format specified in section 2.2.4.2, and silently drop the message if it does not correctly match the format.

2. Extract the objects in the message from the ARRAY_NAME_VALUE field in the NOTIFY_MESSAGE message. Each element in this array is STRUCTURE_NAME_VALUE and represents one object. If the local subscription flag is set, prepare a list of objects to report to the higher-layer protocol or application. If the local subscription flag is set, report the list of objects to the higher-layer protocol or application. Otherwise, discard the list of objects.

3.1.5.6 Receiving a RESPONSE_MESSAGE Message

When a node receives a RESPONSE_MESSAGE message, it MUST perform the following steps:

1. Verify that the RESPONSE_MESSAGE message correctly matches the format specified in section 2.2.4.6, and silently drop the message if it does not correctly match the format.

2. Extract the ARRAY_NAME_VALUE field from the RESPONSE_MESSAGE message. Each element in this array is STRUCTURE_NAME_VALUE and represents one object. If the response conversation flag is set, prepare a list of objects to report to the higher-layer protocol or application.

3. If the response conversation flag is set, report the list of objects to the higher-layer protocol or application and unset the response conversation flag. Otherwise, discard the list of objects.

3.1.5.7 Receiving an APPLICATION_DEFINED_MESSAGE Message

When a node receives an APPLICATION_DEFINED_MESSAGE message, it MUST perform the following steps:

1. Verify that the APPLICATION_DEFINED_MESSAGE message correctly matches the format specified in section 2.2.4.4, and silently drop the message if it does not correctly match the format.

2. Extract the STRING_MIME_TYPE and STRING_VALUE fields from the STRUCTURE_MIME_TYPE_TEXT field body embedded in the message. Report the MIME type and value strings to the higher-layer protocol or application.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

3.1.7.1 Disconnection

In case of unexpected disconnection, both peers MUST delete all state associated with the connection. Reconnection is left to the discretion of the higher-layer protocol or application.

4 Protocol Examples

4.1 Peer Discovery

The application uses some **out of band** means to discover the network endpoint on which the peer's P2PPI implementation is listening. The application supplies the endpoint to the node.

4.2 Initiating an Application Session

An application asks a P2PPI node (hereafter, "the node") to initiate an application session with another node (hereafter, "the peer"). The node, in turn, takes the steps discussed in the following sections.

4.3 Connection, Authentication, and Authorization

The node establishes a TCP connection with its peer. It then secures this connection by using the TLS Protocol version 1.0. The node and the peer authenticate each other during this process.

4.4 Publishing Rich Presence

The peer publishes its rich presence. The user supplies the string "available". The peer adds an object with the name "1d6ccc02-3ec4-453b-b986-470b610cb958" and the value "available" to the published objects list but does not send a NOTIFY_MESSAGE.

4.5 Determining Rich Presence

The application wants to report the rich presence of the peer to the user of the system. The application requests that the node subscribe to the peer. The node transmits a SUBSCRIBE_MESSAGE message to the peer. The peer responds with a NOTIFY_MESSAGE Message that contains a list of objects previously published by the application on the peer's computer.

The node receives the list of objects and reports them to the application. The application examines the object list and finds an object with the name "1d6ccc02-3ec4-453b-b986-470b610cb958". The application understands that this object carries rich presence information. The application reports the value of this object, the string "available", to the user of the system.

4.6 Going Away

The user of the system gets hungry and sets his or her rich presence to the string "out to lunch". The application requests that the node publish a rich presence record having the name "1d6ccc02-3ec4-453b-b986-470b610cb958" and the value "out to lunch". The node adds the object to the published objects list but does not send a NOTIFY_MESSAGE message because it has not received a SUBSCRIBE_MESSAGE message from the peer.

The application does not accept further updates from the peer, because the user of the system has left the console. The application requests that the node unsubscribe. The node sends the peer an UNSUBSCRIBE_MESSAGE message.

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

Security Parameter	Section
Security Protocol	Transport Layer Security (section 2.1.2)
Certificate Format	Certificate Format (section 2.1.2.1)

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Vista® operating system
- Windows® 7 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index