[MS-NFPB-Diff]:

Near Field Proximity: Bidirectional Services Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **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 can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are 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 as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does 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 documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
1/31/2013	1.0	New	Released new document.
8/8/2013	2.0	Major	Significantly changed the technical content.
11/14/2013	3.0	Major	Significantly changed the technical content.
2/13/2014	4.0	Major	Significantly changed the technical content.
5/15/2014	5.0	Major	Significantly changed the technical content.
6/30/2015	6.0	Major	Significantly changed the technical content.
10/16/2015	6.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	7.0	Major	Significantly changed the technical content.
6/1/2017	8.0	Major	Significantly changed the technical content.
12/1/2017	8.0	None	No changes to the meaning, language, or formatting of the technical content.
9/12/2018	9.0	Major	Significantly changed the technical content.
4/7/2021	10.0	Major	Significantly changed the technical content.
6/25/2021	11.0	Major	Significantly changed the technical content.
4/23/2024	12.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

The Near Field Proximity: Bidirectional Services Protocol provides a way for devices such as smartphones to discover services and version information on other devices. It provides a transport-agnostic means of building up impromptu connections between peers, so it can be used on any transport system where peers can subscribe to message types and publish messages based on those types. A prototypical transport is Near Field Communication (NFC) [ECMA-340].

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 (Updated Section) Glossary

This document uses the following terms:

authentication: The ability of one entity to determine the identity of another entity.

base64 encoding: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable ASCII characters, as described in [RFC4648].

big-endian: Multiple-byte values that are byte-ordered with the most significant byte stored in the memory location with the lowest address.

binary large object (BLOB): A discrete packet of data that is stored in a database and is treated as a sequence of uninterpreted bytes.

Bluetooth (BT): A wireless technology standard which is managed by the Bluetooth Special Interest Group and that is used for exchanging data over short distances between mobile and fixed devices.

ChannelID: An 8-byte value used in message exchanges to identify the channel on which the next message is published. It is generated by using cryptographically secure pseudo-random numbers to make the chance of collision in the 64-bit address space unlikely.

domain: A set of users and computers sharing a common namespace and management infrastructure. At least one computer member of the set ~~must~~has to act as a domain controller (DC) and host a member list that identifies all members of the domain, as well as optionally hosting the Active Directory service. The domain controller provides authentication of members, creating a unit of trust for its members. Each domain has an identifier that is shared among its members. For more information, see [MS-AUTHSOD] section 1.1.1.5 and [MS-ADTS].

Elliptic Curve Diffie-Hellman (ECDH): A key agreement protocol that allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel.

encryption: In cryptography, the process of obscuring information to make it unreadable without special knowledge.

Internet Protocol version 4 (IPv4): An Internet protocol that has 32-bit source and destination addresses. IPv4 is the predecessor of IPv6.

Internet Protocol version 6 (IPv6): A revised version of the Internet Protocol (IP) designed to address growth on the Internet. Improvements include a 128-bit IP address size, expanded routing capabilities, and support for authentication and privacy.

KeepAlive timer: A method of tracking the currency of an instance of Session Factory or OOB Connector object in the abstract data model. The timer is started when an object instance is created, and it keeps track of the number of references to that instance by protocol clients. When the number of client references reaches zero, the object is deleted.

key: In cryptography, a generic term used to refer to cryptographic data that is used to initialize a cryptographic algorithm. Keys are also sometimes referred to as keying material.

key derivation: The act of deriving a cryptographic key from another value (for example, the derivation of a cryptographic key from a password).

key exchange: A synonym for key establishment. The procedure that results in shared secret keying material among different parties. Key agreement and key transport are two forms of key exchange. For more information, see [CRYPTO] section 1.11, [SP800-56A] section 3.1, and [IEEE1363] section 3.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

Media Access Control (MAC) address: A hardware address provided by the network interface vendor that uniquely identifies each interface on a physical network for communication with other interfaces, as specified in [IEEE802.3]. It is used by the media access control sublayer of the data link layer of a network connection.

Near Field Communication (NFC): An international standard for short-range wireless, contactless connectivity that provides intuitive, simple, and safe communication between electronic devices. NFC is the technology on smartphones that makes proximity scenarios possible. For example, it allows a user to wave the smartphone over a NFC-compatible device to send information without needing to touch the devices together or go through multiple steps setting up a connection.

network layer (L3): The third layer in the ISO/OSI reference model that provides the ability to transfer variable length data sequences from a source host on one network to a destination host on a different network while maintaining the quality of service (QoS) requested by the transport layer.

organizationally unique identifier (OUI): A unique 24-bit string that uniquely identifies a vendor, manufacturer, or organization on a worldwide l basis, as specified in [IEEE-OUI]. The OUI is used to help distinguish both physical devices and software, such as a network protocol, that belong to one entity from those that belong to another.

out-of-band (OOB): A process for authenticating a user where two communication channels are used simultaneously between two devices or roles. A cellular network is an example of a channel that is commonly used for performing out-of-band authentication.

peer-to-peer: A server-less networking technology that allows several participating network devices to share resources and communicate directly with each other.

private key: One of a pair of keys used in public-key cryptography. The private key is kept secret and is used to decrypt data that has been encrypted with the corresponding public key. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

pub/sub: Refers to publication/subscription, a design model in which publishers send notification of events that are received by subscribers, which have registered for those events.

public key: One of a pair of keys used in public-key cryptography. The public key is distributed freely and published as part of a digital certificate. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

publication: A message placed on the underlying transport along with a type identifier. Multiple individual publication messages may be placed on the transport with the same type identifier. Peers receive a published message if they have previously subscribed to it by type.

radio frequency communications (RFCOMM): A protocol that provides serial port emulation of EIA-232 (formerly RS-232) control signals over the Bluetooth baseband layer. RFCOMM is used to create a virtual serial data stream to enable binary data transport.

subscription: A registration performed by a subscriber to specify a requirement to receive events, future messages, or historical data.

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

universally unique identifier (UUID): A 128-bit value. UUIDs can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects in cross-process communication such as client and server interfaces, manager entry-point vectors, and RPC objects. UUIDs are highly likely to be unique. UUIDs are also known as globally unique identifiers (GUIDs) and these terms are used interchangeably in the Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the UUID. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the UUID.

UTF-8: A byte-oriented standard for encoding Unicode characters, defined in the Unicode standard. Unless specified otherwise, this term refers to the UTF-8 encoding form specified in [UNICODE5.0.0/2007] section 3.9.

Wi-Fi Direct (WFD): A standard that allows Wi-Fi devices to connect to each other without requiring a wireless access point (WAP). This standard enables WFD devices to transfer data directly among each other resulting in significant reductions in setup.

winning peer: A peer that has the preference to be the server in future message exchanges. This term applies to the relevant message exchange between the two peers.

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 (Updated Section) 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.

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

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

[WF-P2P1.2] Wi-Fi Alliance, "Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.2", https://www.wi-fi.org/wi-fi-peer-to-peer-p2p-technical-specification-v12

Note There is a charge to download the specification.

[WF-WSC2.0.2] Wi-Fi Alliance, "Wi-Fi Simple Configuration Technical Specification v2.0.2", August 2011, https://www.wi-fi.org/wi-fi-simple-configuration-technical-specification-v202

Note There is a charge to download the specification.

1.2.2 (Updated Section) Informative References

[ECMA-340] ECMA International, "Near Field Communication Interface and Protocol (NFCIP-1)", 2nd edition, ECMA-340, December 2004, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-340.pdf

[IEEE-OUI] IEEE Standards Association, "IEEE MAC Address Block Large (MA-L) Field Registration Authority Public Listing", http://standards-oui.ieee.org/oui/oui.txt

[MS-NFPS] Microsoft Corporation, "Near Field Proximity: Sharing Protocol".

[NSA] National Security Agency, "Suite B Implementer's Guide to FIPS 186-3 (ECDSA)", February 2010, https://apps.nsa.gov/iaarchive/library/ia-guidance/ia-solutions-for-classified/algorithm-guidancesilo.tips/download/suite-b-implementersimplementer-s-guide-to-fips-186-3-ecdsa.cfm

[RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, February 2006, httphttps://www.ietfrfc-editor.org/rfcinfo/rfc4380.txt

1.3 Overview

Although the underlying transport for the Near Field Proximity: Bidirectional Services Protocol is undefined, the protocol models the transport as a publication/subscription system to exchange messages between peers. The transport is modeled with the assumption that it is either active or inactive. Typically, there is user-intent to activate the transport, but that is not required. When active, the transport transmits all local publications to peer subscribers on the other side of the transport, and it does so just once. When inactive, the transport does not transmit or receive any data.

Peers can use the information in protocol messages to activate services. A **Service Descriptor** contains a list of services and versions that are defined by this protocol. Each service is identified by a UUID, which peers can use to send activation messages for the service. To exchange **Service Descriptor** messages, each peer both publishes and subscribes to the service.

Some services can be inherently client/server, so that upon reception of a **Service Descriptor** message with a compatible service, a client can immediately activate this service by replying with an activation message.

Other services can be inherently peer-to-peer and use client preference ID fields to determine the specific peer that will begin the next phase of service activation. Each peer randomly generates an ID to provide highly probable uniqueness and includes the ID in each **Service Descriptor**. The peer that generates the numerically higher ID is the winning peer, and it sends the next message for a given service. A specific peer can only be winning or losing with respect to another specific peer.

When a client application establishes a connection over the network layer (L3), it validates the connection by exchanging handshake data with the server, as described in section 1.3.4 Connection Validation.

1.3.1 Session Factory Service Activation

Either peer can choose to activate the remote peer's **Session Factory** service in order to establish a single-instanced session between an application running locally and another instance of the same application running on the remote peer. Optionally, a peer can support launching or acquiring the application in addition to, or instead of, establishing the single-instanced session between two instances of the application.

1.3.2 OOB Connector Service Activation

A winning peer can activate the remote peer's **OOB Connector** service in order to provide out-of-band (OOB) transport options for the peers to connect. These connection options allow the **Session Factory** service a simple means of address resolution.

1.3.3 Session Activation

The following diagram shows a generic sequence of session activation.

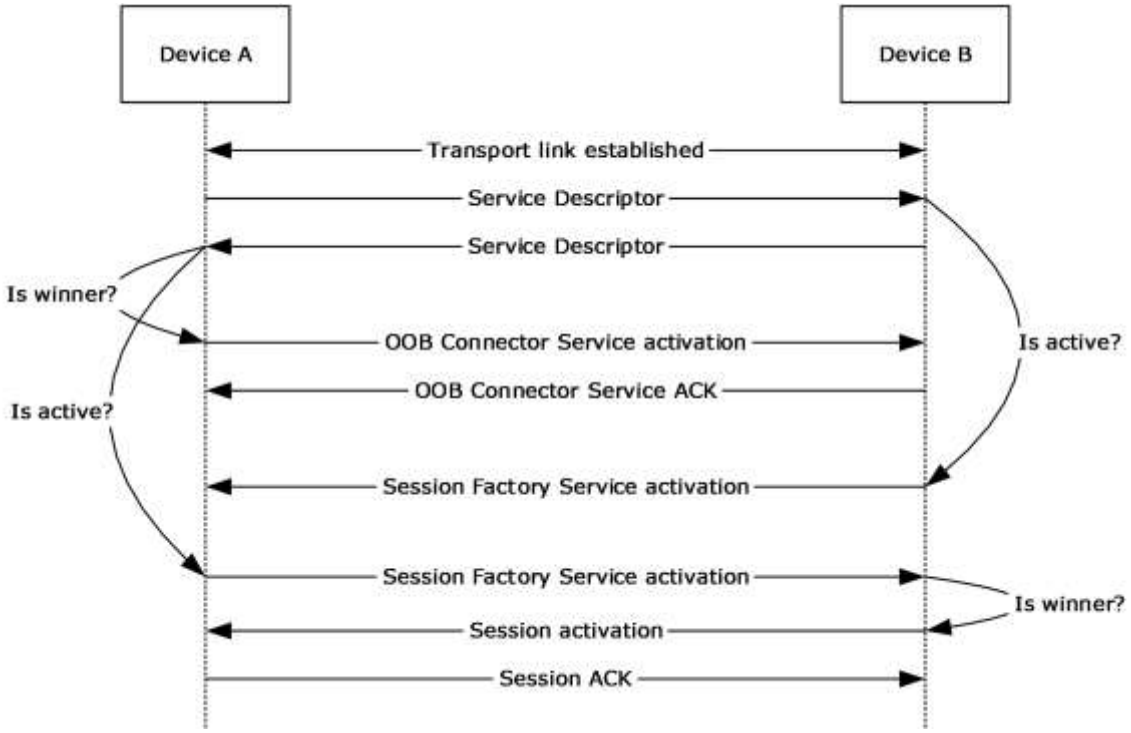


Figure 1: Session activation sequence

1.3.4 Connection Validation

After session activation (section 1.3.3) and subsequent L3 establishment, the client validates the connection by exchanging handshake data with the server, as shown in the following diagram. The handshake data consists of an Accept Header (section 2.2.1).

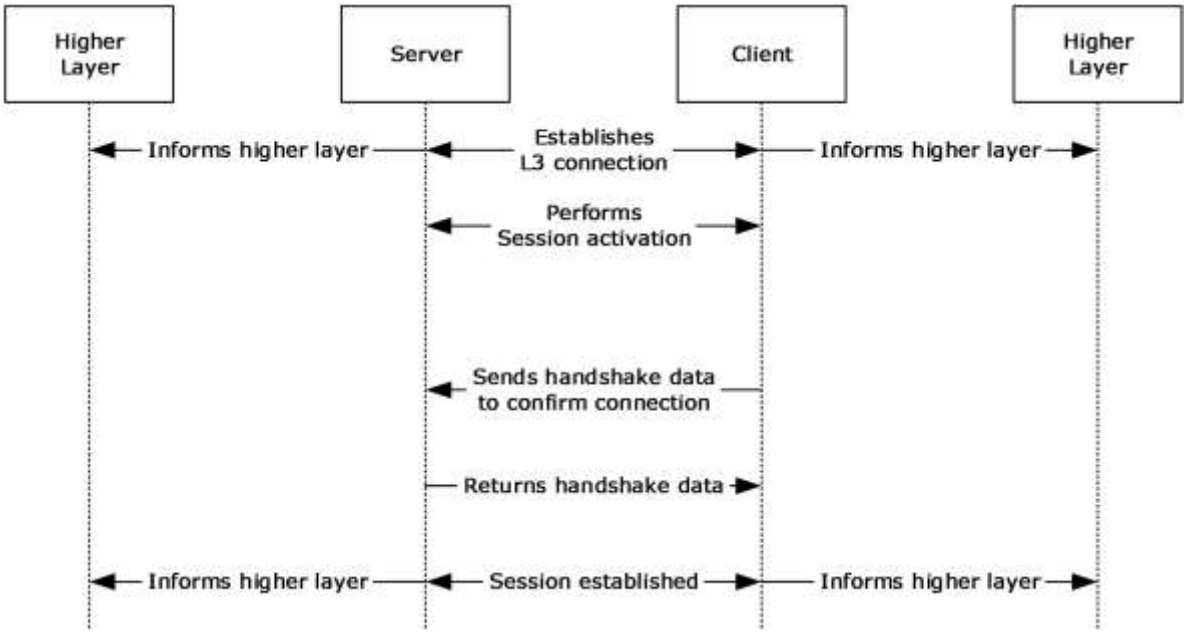


Figure 2: Connection validation

1.4 Relationship to Other Protocols

The following diagram shows the relationship of the Near Field Proximity: Bidirectional Services Protocol with other protocols.

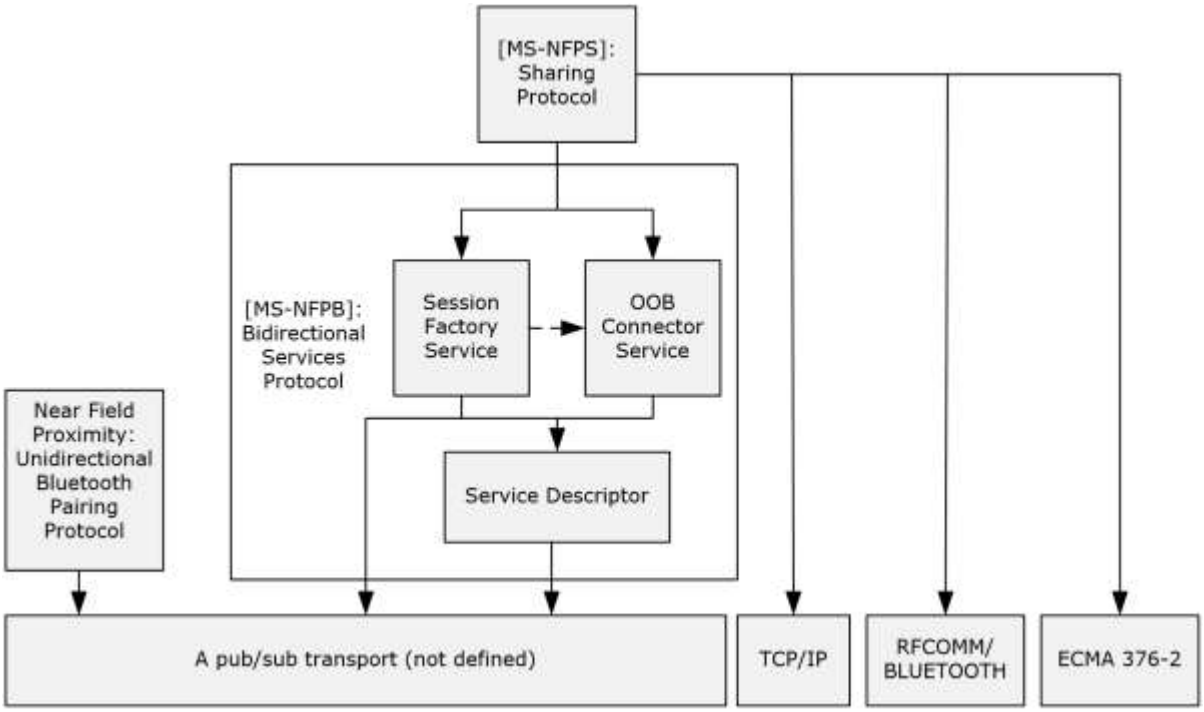


Figure 3: Relationship to other protocols

The **Service Descriptor** does not depend on any specific transport, and as such, does not technically depend on any other protocol.

The **OOB Connector** service depends only on the **Service Descriptor** for service discovery, versioning, and winner/loser role-determination. The **OOB Connector** service uses the publication/subscription transport for activation and acknowledgment. The exchanged data is combined with data exchanged by the **Session Factory** service by higher-level protocols to establish single-instanced connections between applications.

The **Session Factory** service depends only on the **Service Descriptor** for service discovery and versioning; it does not depend on the **Service Descriptor** for winner/loser role-determination. The **Session Factory** service performs winner/loser role-determination in the activation message exchange rather than the **Service Descriptor** message exchange. The **Session Factory** service uses the publication/subscription transport for winner/loser role-determination, activation, and acknowledgement. As described previously, the exchanged data in activation and acknowledgment scenarios is combined with data exchanged by the **OOB Connector** service by higher-level protocols to establish single-instanced connections between applications. These connections are typically established over TCP/IP or RFCOMM/Bluetooth. However, the **OOB Connector** service does not mandate a specific transport. The higher-level protocol can determine, at runtime, which transports to use to establish the session's connection.

The Near Field Proximity: Sharing Protocol [MS-NFPS] is an example of a higher-level protocol.

1.5 Prerequisites/Preconditions

Peers communicate by using compatible networking technologies such as TCP/IP over wireless networks. There are no other preconditions or prerequisites for this protocol to function between peers. There are no presupposed security associations or connections required between peers except those that are required by the unspecified pub/sub transport link layer.

1.6 Applicability Statement

The Near Field Proximity: Bidirectional Services Protocol is well-suited to function on top of transports such as Near Field Communication (NFC) [ECMA-340]. This protocol has been designed for linking two applications for the purposes of simple real-time sharing of files. This protocol is designed to function in cross-platform, cross-domain, and non-domain configurations.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Security and Authentication Methods:** The **Service Descriptor** neither requires nor provides any security or authentication methods. The **OOB Connector** service and **Session Factory** service contain specific embedded key exchange algorithms that can be used by higher-level protocols to provide a level of security for continuing communication over an OOB channel. However, the specific algorithms are attached to specific versions of the **Session Factory** service; there is not a more granular means of negotiating algorithms.
- **Capability Negotiation:** This protocol performs explicit capability negotiation by using the **Service Descriptor** structure (section 2.2.9).

1.8 Vendor-Extensible Fields

1.8.1 Service Descriptor Entries

Each entry within a **Service Descriptor** message (section 2.2.8) is a **Service Descriptor** structure (section 2.2.9), which contains a service activation UUID that uniquely identifies each service. Vendors that wish to define a new service MUST generate a new UUID for the service.

1.8.2 AppInfo Platform Qualifiers

AppInfo platform qualifiers SHOULD be defined by each vendor that implements any of the following protocols:

- The Session Factory protocol
- The Launch App protocol
- The Launch Compatible App protocol

A vendor SHOULD define its qualifier based on a domain name, like "fabrikam.com", which is owned by the vendor, to ensure that no other vendor uses the same value. The platform qualifier is specified by an **AppInfo** structure (section 2.2.2).

1.8.3 Session Activation and Acknowledgment Extensions

The Session Factory protocol has an extension pattern that can be used by implementations of this protocol. If an implementation defines an extension, a random 8-byte value SHOULD be used to ensure that no other implementation uses the same value. 8 bytes is enough to make collisions unlikely.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

As stated earlier in this document, a specific transport is not defined. However, the general requirements for a transport are as follows:

Transports on which this protocol is built MUST be able to provide reliable packet-based delivery of messages. The transport MUST be able to provide the size of each message, independently of its payload, to the component that implements the protocol.

Messages are published and subscribed over the transport on channels that are analogous to ports in TCP/IP. Well-known channel names allow two peers to establish initial communication. If a bi-directional exchange is required, the first message SHOULD contain an ID that allows the receiver to return a message on a channel based on that ID.

In this document, all multi-message exchanges except the final message use an 8-byte identifier to denote the channel on which the following message MUST be published. This identifier is referred to as a ChannelID. ChannelIDs MUST be generated by using cryptographically secure pseudo-random numbers to reduce the likelihood of collisions. A collision can result in a protocol failure, which means that the exchange MUST be manually attempted again. With Near Field Communication (NFC) as the typical transport, the user is required to tap or swipe the devices together again. However, with 64 bits of ChannelID address space, the chance of collision is unlikely.

Some transports, like Near Field Communication (NFC), require that the channel be encoded in characters that are allowed in a URI. When that is the case, the channel MUST be encoded by using base64 [RFC2045], with the exception that padding characters MUST be omitted. ChannelIDs that are 8 bytes therefore result in channels that are exactly 11 characters long.

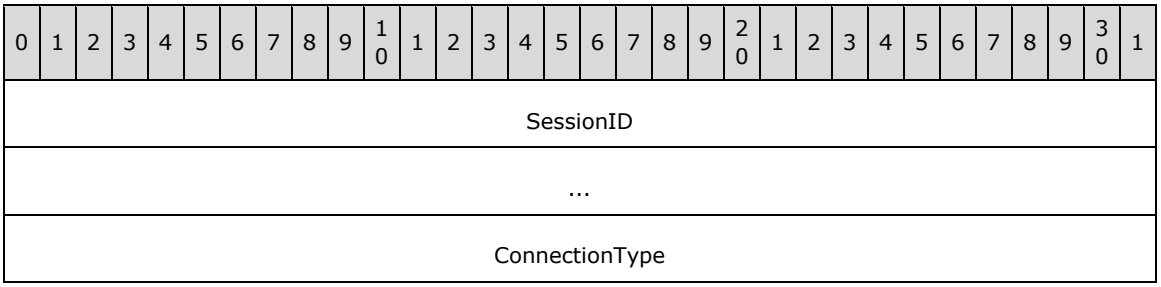
2.2 Message Syntax

None of the messages in this protocol has alignment requirements; that is, there are no padding bytes for forcing specific alignment. Additionally, fields are made as small as possible to optimize for fast transmission over low-bit-rate transports. Unless explicitly specified otherwise, all fields use big-endian encoding.

There is no single common header for all messages in this protocol; however, there are some common structures within messages, which are described in the sections that follow.

2.2.1 Accept Header

The **Accept** header is sent by a client application to the server after a session is activated over an L3 connection. The server MUST validate the handshake data and return the same **Accept** header to the client on the connected socket.



SessionID (8 bytes): This MUST be the same value that the client generated and sent in the **ReplyChannelID** field of the preceding **Session Activation** message (section 2.2.11). The

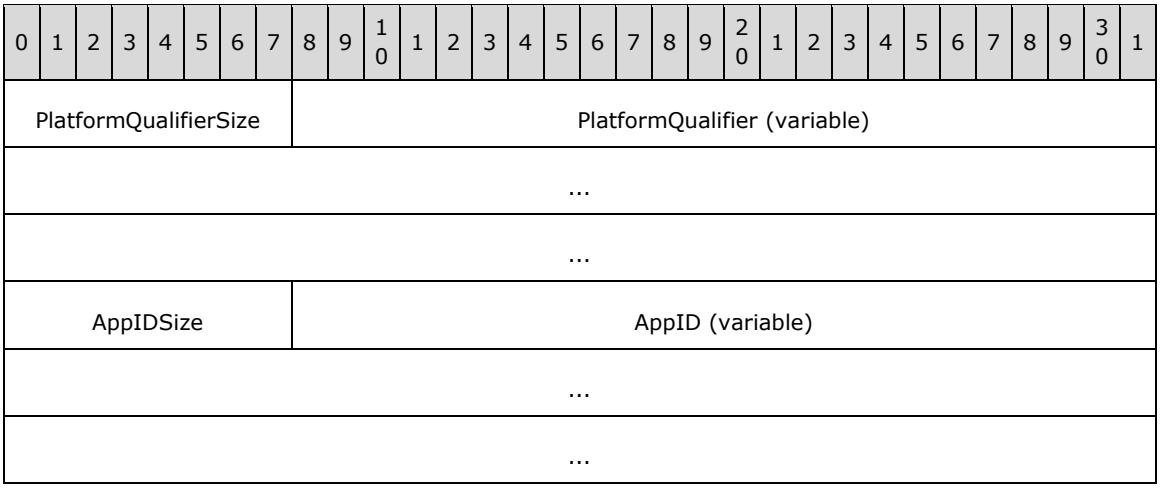
SessionID verification ensures that the applications that tapped are the ones that are connected over L3.

ConnectionType (4 bytes): Indicates the type of transport that the server and client connected over. This MUST be set to one of the following values.

Value	Connection Type
0x00000000	Wi-Fi Direct (WFD)
0x00000001	Link Local (IPv6)
0x00000002	Link Local (IPv4)
0x00000004	Bluetooth

2.2.2 AppInfo Structure

The **AppInfo** structure is used by platforms implementing the Session Factory protocol, the Launch App protocol, or the Launch Compatible App protocol. The **AppInfo** structure format is specified as follows.



PlatformQualifierSize (1 byte): The length of the **PlatformQualifier** field, in bytes. The value MUST be greater than zero and less than or equal to 20.

PlatformQualifier (variable): A UTF-8 string that specifies the namespace for the application identifier **AppID**. This usually refers to an application store or application environment within an OS platform. The string MUST NOT be null-terminated or contain embedded nulls.

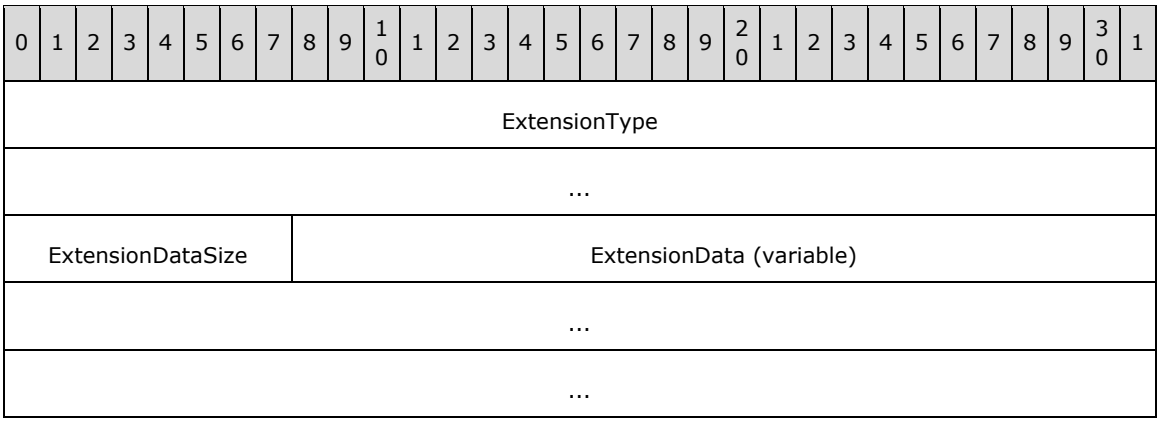
AppIDSize (1 byte): The length of the **AppID** field, in bytes. The value MUST be nonzero.

AppID (variable): A platform-dependent identifier for a specific application. Platforms SHOULD use the smallest identifier size that is practical in order to produce compact designs. This field contains arbitrary binary data up to the length specified in the **AppIDSize** field.

A message containing an **AppInfo** structure that does not meet any of the preceding field criteria MUST be ignored.

2.2.3 Extension Structure

The **Extension** structure can be used by platforms implementing the Session Factory protocol. The **Extension** structure format is specified as follows.



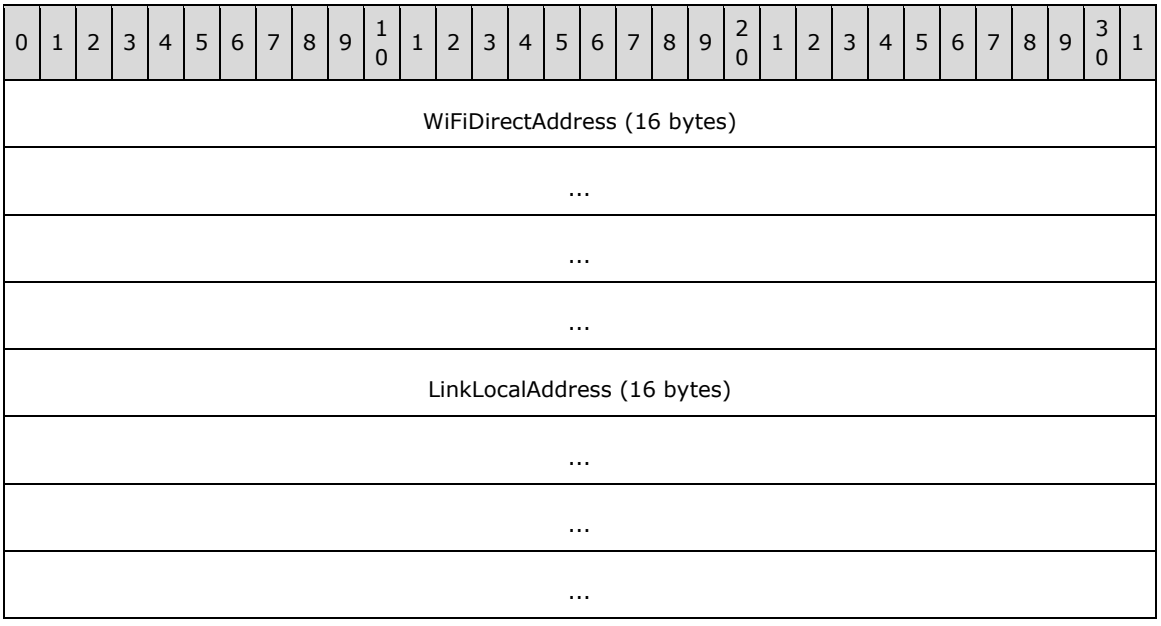
ExtensionType (8 bytes): A platform-dependent value that identifies the type of the extension. A platform that defines an extension SHOULD declare and publish a random number to identify that extension.

ExtensionDataSize (1 byte): The length of the **ExtensionData** field in bytes. The value MUST be nonzero. Extensions not meeting this criterion MUST be ignored.

ExtensionData (variable): A platform-dependent BLOB of data for a specific extension. Platforms SHOULD use the smallest data size that is practical in order to produce compact designs. This field contains arbitrary binary data up to the length specified in the **ExtensionDataSize** field.

2.2.4 OOB Connector Service ACK Message

The **OOB Connector Service ACK** message is the acknowledgment reply to the **OOB Connector Service Activation** message (section 2.2.5). The **OOB Connector Service ACK** message format is specified as follows.



IPv4LinkLocalAddress (16 bytes)	
...	
...	
...	
ProximityAddress (16 bytes)	
...	
...	
...	
GlobalAddress (16 bytes)	
...	
...	
...	
TeredoAddress (16 bytes)	
...	
...	
...	
BlueToothMACAddress	
...	
WiFiDirectListenBlobLength	WiFiDirectListenBlob (variable, optional)
...	
...	

WiFiDirectAddress (16 bytes): A randomly-generated IPv6 link-local address. If the OOB Connector protocol results in a new WFD layer 2 link, the publisher MUST assign this address to the link in order to allow layer 3 connectivity.

Use of this value is optional. It SHOULD be set to zero if not used.

LinkLocalAddress (16 bytes): The best link-local IPv6 address assigned to the publisher. "Best" is defined in order of decreasing precedence of the following: connectivity, Wi-Fi infrastructure links,

transmit bit rate, receive bit rate, and non-tunnel links. If no link-local address is suitable, the value of this field SHOULD be zero.

IPv4LinkLocalAddress (16 bytes): The best IPv4 link-local address assigned to the publisher in V4-MAPPED format. "Best" is defined in order of decreasing precedence of the following: connectivity, Wi-Fi infrastructure links, transmit bit rate, receive bit rate, and non-tunnel links. If no IPv4 link-local address is suitable, the value of this field SHOULD be zero.

This value provides for connectivity over networks that do not support link-local IPv6 traffic, such as some legacy Wi-Fi networks.

ProximityAddress (16 bytes): An IPv6 address assigned to the transport link that this message is published on. Not all pub/sub transports support IP connectivity; an example that can support IP is TransferJet. If the underlying transport does not support IP, the value of this field SHOULD be zero.

GlobalAddress (16 bytes): The "best" global IPv6 address assigned to the publisher. "Best" is defined in order of decreasing precedence of the following: connectivity, non-Teredo-type [RFC4380], transmit bit rate, receive bit rate, and, non-tunnel links. If no global address is suitable, the value of this field SHOULD be zero.

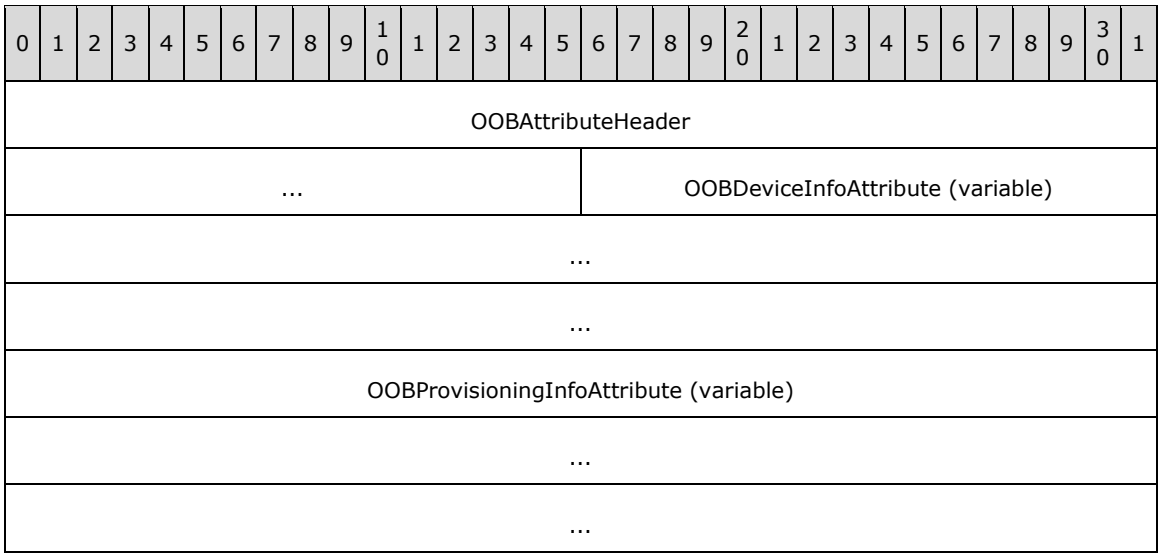
TeredoAddress (16 bytes): The"best" Teredo-type IPv6 address assigned to the publisher. "Best" is defined in order of decreasing precedence of the following: connectivity, Teredo-type, transmit bitrate, and receive bitrate. If no Teredo address is suitable, the value of this field SHOULD be zero.

The Teredo Tunneling protocol is a technology that allows Internet nodes to have global IPv6 addressing capability tunneled over IPv4 networks.

BlueToothMACAddress (8 bytes): The Media Access Control (MAC) address of the best Bluetooth adapter available to the publisher. "Best" is defined by the platform; many platforms only allow zero or one Bluetooth adapter. If no Bluetooth adapter is available, the value of this field SHOULD be zero.

WiFiDirectListenBlobLength (2 bytes): The length, in bytes, of the **WiFiDirectListenBlob** field that follows. If the value of this field is zero, the **WiFiDirectListenBlob** field is not present.

WiFiDirectListenBlob (variable, optional): The WFD listen data, in the following format. All values for this structure are in little-endian format, unless specified otherwise.



OOBConfigurationTimeoutAttribute

OOBAttributeHeader (6 bytes): The OOB Attribute header (section 2.2.4.1).

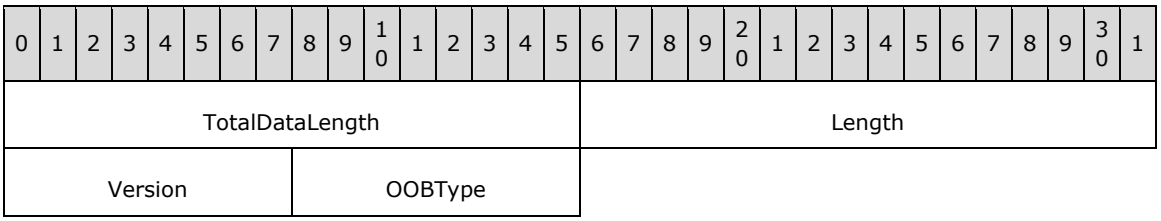
OOBDeviceInfoAttribute (variable): OOB data in Device Info Attribute format (section 2.2.4.4).

OOBProvisioningInfoAttribute (variable): OOB data in Provisioning Info Attribute format (section 2.2.4.5).

OOBConfigurationTimeoutAttribute (4 bytes): OOB data in Configuration Timeout Attribute format (section 2.2.4.6).

2.2.4.1 OOB Attribute Header

The **OOB Attribute** header defines the version and size of either the **WiFiDirectListenBlob** in an **OOB Connector Service ACK** message (section 2.2.4) or the **WiFiDirectConnectBlob** in an **OOB Connector Service Activation** message (section 2.2.5). The **OOB Attribute** header is specified as follows.



TotalDataLength (2 bytes): The length, in bytes, of the OOB data BLOB, which can be either a WiFiDirectListenBlob or a WiFiDirectConnectBlob, depending on the value of the **OOBType** field.

Length (2 bytes): The length, in bytes, of the following fields.

Version (1 byte): A value identifying the version of OOB data. This value MUST be 0x10.

OOBType (1 byte): A value identifying the type of OOB data. This value MUST be one of the following.

OOB type	Description
0x01	OOB provisioning listener data
0x02	OOB provisioning connector data

2.2.4.2 OOB Attribute Type Constants

The **OOB Attribute Type** constants specify the identifiers of possible formats of OOB attribute data.

Attribute ID	Attribute type
0	OOB status
1	OOB device info
2	OOB provisioning info

Attribute ID	Attribute type
3	OOB group ID
4	OOB listen channel
5	OOB configuration timeout
6-220	Reserved

2.2.4.3 OOB Provisioning Settings Constants

The **OOB Provisioning Setting**s constants specify the bit settings of possible provisioning options for the **OOB Provisioning Info Attribute** format (section 2.2.4.5).

Bit(s)	Meaning
0	Create new group
1	Enforce group type setting
2	Desired group type
3-7	Reserved

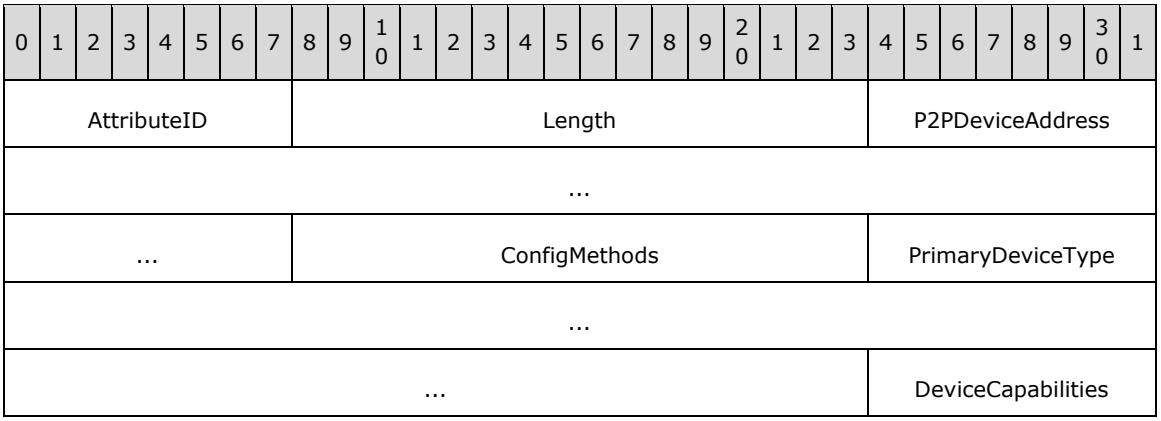
Bit 0: This bit is set to 1 if the provisioning information can be used for forming a new group with the target peer-to-peer device; otherwise, the information is used for joining an existing group.

Bit 1: This bit is set to 1 to enforce the desired group type setting in **Bit 2**; otherwise, the desired group type setting is simply a preference.

Bit 2: This bit is set to 0 if the desired group type is transient and set to 1 if the desired group type is persistent.

2.2.4.4 OOB Device Info Attribute Format

The **OOB Device Info Attribute** format defines device information in either the WiFiDirectListenBlob in an OOB Connector Service ACK message (section 2.2.4) or the WiFiDirectConnectBlob in an OOB Connector Service Activation message (section 2.2.5). The **OOB Device Info Attribute Format** is specified as follows.



DeviceName (variable)
...
...

AttributeID (1 byte): The type of OOB attribute, as defined in section 2.2.4.2. This value is 0x01 for the **OOB Device Info Attribute** format.

Length (2 bytes): The length, in bytes, of the following fields.

P2PDeviceAddress (6 bytes): An identifier that uniquely references a peer-to-peer device [WF-P2P1.2].

ConfigMethods (2 bytes): The Wi-Fi Simple Configuration (WSC) methods [WF-WSC2.0.2] that are supported by this device. Byte ordering within the **ConfigMethods** field is big-endian.

PrimaryDeviceType (8 bytes): The primary device type of the peer-to-peer device in the following format. Byte ordering within the **PrimaryDeviceType** field is big-endian.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
CategoryID																OUI															
...																SubcategoryID															

CategoryID (2 bytes): The vendor-independent main device category identifier. The predefined values for this field and the corresponding values for the **SubcategoryID** field are shown in the following table. Note that there is no way to indicate a vendor-specific main device category. The organizationally unique identifier (OUI) [IEEE-OUI] value in the **OUI** field applies only to the interpretation of the subcategory value.

Main device category	CategoryID	Device subcategory	SubcategoryID
Computer	1	PC	1
		Server	2
		Media Center	3
Input Device	2		
Printers, Scanners, Faxes, and Copiers	3	Printer	1
		Scanner	2
Camera	4	Digital Still Camera	1
Storage	5	NAS	1
Network Infrastructure	6	Access point	1
		Router	2
		Switch	3
Displays	7	Television	1

Main device category	CategoryID	Device subcategory	SubcategoryID
		Electronic Picture Frame	2
		Projector	3
Multimedia Devices	8	DAR	1
		PVR	2
		MCX	3
		DMR	4
Gaming Devices	9	Xbox	1
		Xbox360	2
		Playstation	3
Telephone	10	Windows Mobile	1

OUI (4 bytes): The OUI of the device. For the predefined values specified in the **CategoryID** field, the Wi-Fi Alliance byte values 0x00 0x50 0xF2 0x04 are used.

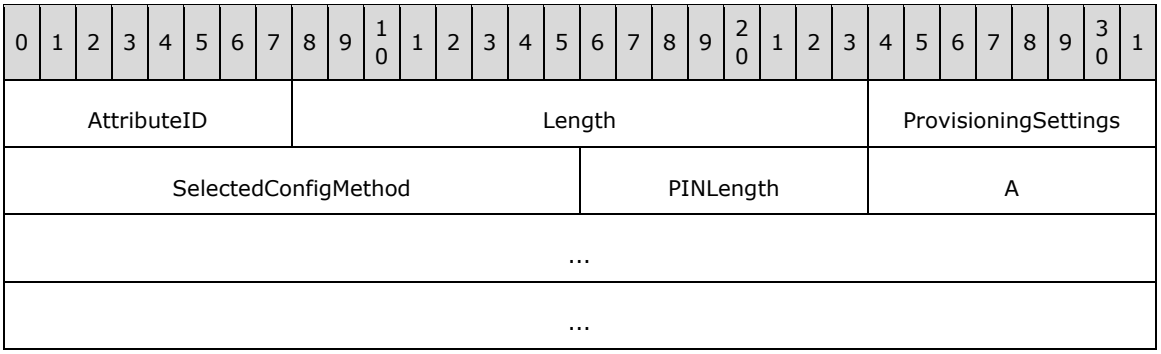
SubcategoryID (2 bytes): The vendor-specific device subcategory identifier. The predefined values for this field are specified in the preceding table.

DeviceCapabilities (1 byte): The capabilities of the peer-to-peer device.

DeviceName (variable): A UTF-8 string that specifies the friendly name of the peer-to-peer device. Byte ordering within the **DeviceName** field is big-endian.

2.2.4.5 OOB Provisioning Info Attribute Format

The **OOB Provisioning Info Attribute** format defines provisioning settings in the WiFiDirectListenBlob in an OOB Connector Service ACK message (section 2.2.4). The **OOB Provisioning Info Attribute** format is specified as follows.



AttributeID (1 byte): The type of OOB attribute, as defined in section 2.2.4.2. This value is 0x02 for the **OOB Provisioning Info Attribute** format.

Length (2 bytes): The length, in bytes, of the following fields.

ProvisioningSettings (1 byte): One or more provisioning bit settings, as defined in section 2.2.4.4.

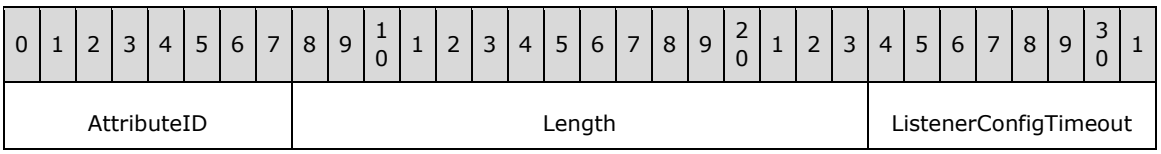
SelectedConfigMethod (2 bytes): The Wi-Fi Simple Configuration (WSC) method [WF-WSC2.0.2] that was selected by a peer-to-peer device for provisioning.

PINLength (1 byte): The length, in bytes, of the following **PINData** field. This field contains a value from 0 to 8 bytes. If it is zero, the **PINData** field is not present.

A - PINData (variable, optional): An array of bytes that represent a PIN to be used for provisioning.

2.2.4.6 OOB Configuration Timeout Attribute Format

The **OOB Configuration Timeout Attribute** format defines the listener timeout in the WiFiDirectListenBlob in an OOB Connector Service ACK message (section 2.2.4). The **OOB Configuration Timeout Attribute** format is specified as follows.



AttributeID (1 byte): The type of OOB attribute, as defined in section 2.2.4.2. This value is 0x05 for OOB Configuration Timeout Attribute format.

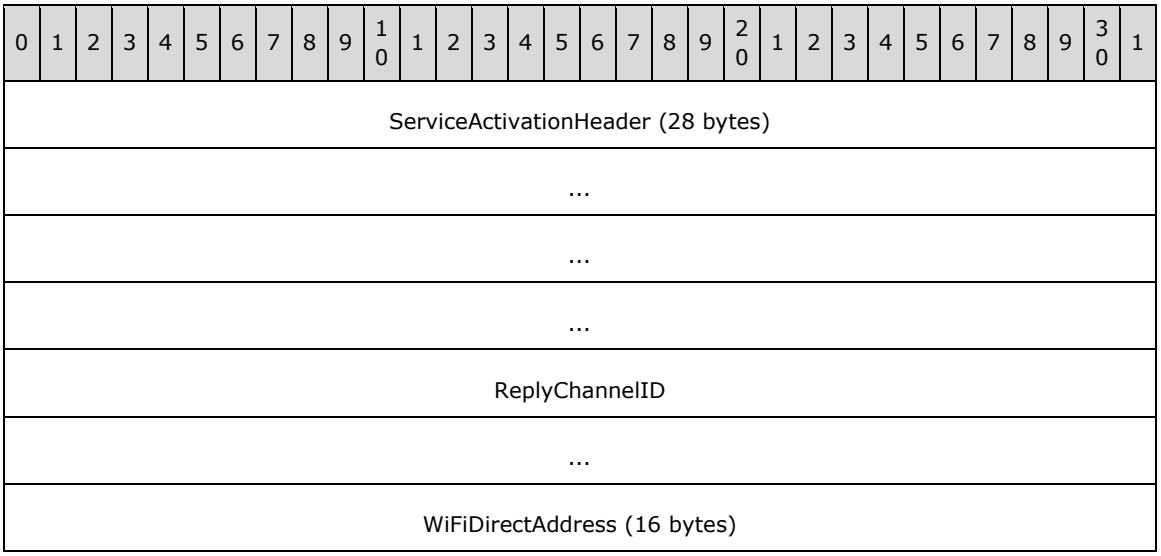
Length (2 bytes): Contains the length of the following fields in the attribute in bytes. This value MUST be 1.

ListenerConfigTimeout (1 byte): The amount of time, in units of 100 milliseconds, this peer-to-peer device will spend waiting for WFD communication after an OOB data transfer. Valid timeout values range from zero to 255.

2.2.5 OOB Connector Service Activation Message

The **OOB Connector Service Activation** message is a reply to the **Service Descriptor** message (section 2.2.8). It is used to establish a paired set of **OOB Connector** objects (section 3.1.1.2) between two peers. If the local **SourceID** is greater than the **ActivationChannelID** in the received **Service Descriptor** message, an **OOB Connector Service Activation** message MUST be published on the **ActivationChannelID** of the **Session Activation** message (section 2.2.11), unless the **OOB Connector** object for that remote service is already created and active.

The **OOB Connector Service Activation** message format is specified as follows.



...
...
...
LinkLocalAddress (16 bytes)
...
...
...
IPv4LinkLocalAddress (16 bytes)
...
...
...
ProximityAddress (16 bytes)
...
...
...
GlobalAddress (16 bytes)
...
...
...
TeredoAddress (16 bytes)
...
...
...
Reserved
BlueToothMACAddress

...	
WiFiDirectConnectBlobLength	WiFiDirectConnectBlob (variable, optional)
...	
...	

ServiceActivationHeader (28 bytes): A **Service Activation** header (section 2.2.7). The **ServiceActivationUUID** MUST be {E46EDA50-9B5D-41F1-B89E-327B5EA38B16}. The **ServiceVersion** MUST be 1.

ReplyChannelID (8 bytes): This value is also the identifier of the **OOB Connector** object (section 3.1.1.2). It MUST be generated at random by the publisher. The publisher of this message MUST subscribe to the **ReplyChannelID** prior to publishing this message to ensure that replies are not missed.

WiFiDirectAddress (16 bytes): A randomly generated IPv6 link-local address. If the OOB Connector protocol results in a new WFD layer 2 link, the publisher MUST assign this address to the link in order to allow layer 3 connectivity.

Use of this field is OPTIONAL. It MAY be ignored and the value SHOULD be set to zero if unused.

LinkLocalAddress (16 bytes): The best link-local IPv6 address assigned to the publisher. "Best" is defined in order of decreasing precedence of the following: connectivity, Wi-Fi infrastructure links, transmit bit rate, receive bit rate, and non-tunnel links. If no link-local address is suitable, the value of this field SHOULD be zero.

IPv4LinkLocalAddress (16 bytes): The best IPv4 link-local address assigned to the publisher in V4-mapped format. "Best" is defined in order of decreasing precedence of the following: connectivity, Wi-Fi infrastructure links, transmit bit rate, receive bit rate, and non-tunnel links. If no IPv4 link-local address is suitable, the value of this field SHOULD be zero.

This value provides for connectivity over networks that do not support link-local IPv6 traffic, such as some legacy Wi-Fi networks.

ProximityAddress (16 bytes): The IPv6 address assigned to the transport link that this message is published on. Not all pub/sub transports support IP connectivity; an example that can support IP is TransferJet. If the underlying transport does not support IP, the value of this field SHOULD be zero.

GlobalAddress (16 bytes): The best global IPv6 address assigned to the publisher. "Best" is defined in order of decreasing precedence of the following: connectivity, non-Teredo-type [RFC4380], transmit bit rate, receive bit rate, and non-tunnel links. If no global address is suitable, the value of this field SHOULD be zero.

TeredoAddress (16 bytes): The best Teredo-type IPv6 address assigned to the publisher. "Best" is defined in order of decreasing precedence of the following: connectivity, Teredo-type, transmit bit rate, and receive bit rate. If no Teredo address is suitable, the value of this field SHOULD be zero.

The Teredo Tunneling protocol is a technology that allows Internet nodes to have global IPv6 addressing capability tunneled over IPv4 networks.

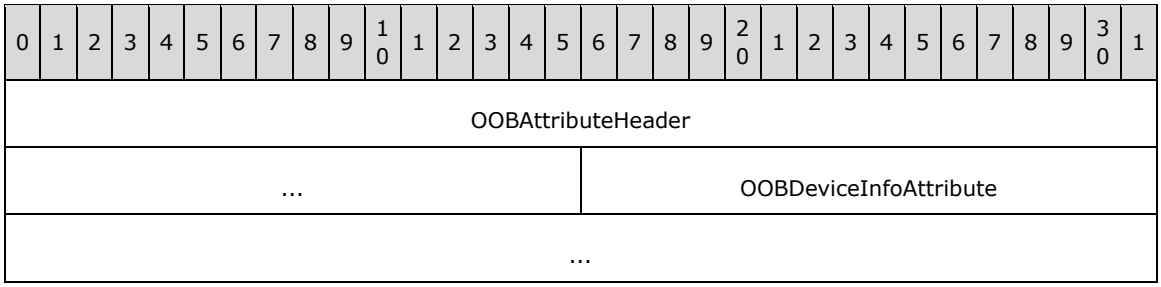
Reserved (4 bytes): This field MUST be set to zero when sent and MUST be ignored when received.

BlueToothMACAddress (8 bytes): The Media Access Control (MAC) address of the best Bluetooth adapter available to the publisher. "Best" is defined by the platform; many platforms only allow

zero or one Bluetooth adapter. If no Bluetooth adapter is available, the value of this field SHOULD be zero.

WiFiDirectConnectBlobLength (2 bytes): The length, in bytes, of the **WiFiDirectConnectBlob** field that follows. If the value of this field is zero, the **WiFiDirectConnectBlob** field is not present.

WiFiDirectConnectBlob (variable, optional): The WFD connect data, in the following format. All values for this structure are in little-endian format, unless specified otherwise.



OOBAttributeHeader (6 bytes): The OOB Attribute header (section 2.2.4.1).

OOBDeviceInfoAttribute (variable): OOB data in Device Info Attribute format (section 2.2.4.4).

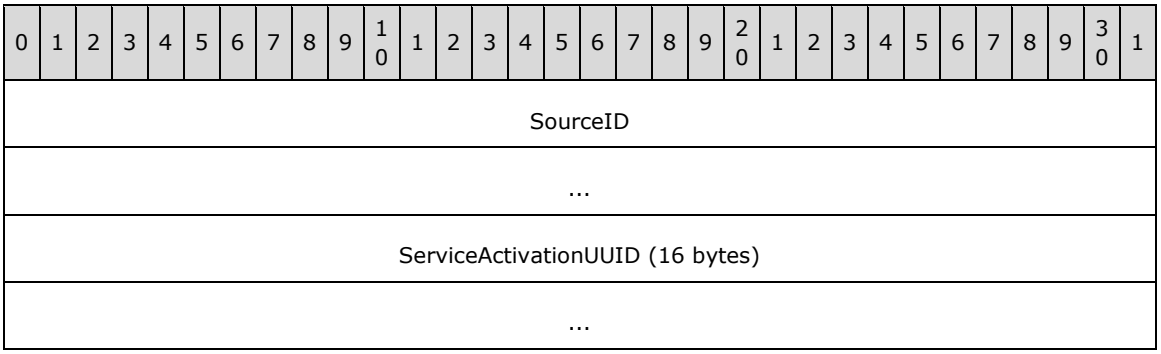
2.2.6 Role Compatibility Constants

The **Role Compatibility** constants SHOULD<1> be used to check the value of the **Role** field in a **Session Factory Service Activation** message (section 2.2.12) for compatibility with the role of the receiver of that message.

Value in Role field	Compatible role value
0x01 (peer role)	0x01 (peer role)
0x02 (host role)	0x03 (client role)
0x03 (client role)	0x02 (host role)

2.2.7 Service Activation Header

The **Service Activation** header is common to all service activation messages. The **Service Activation** header format is specified as follows.



...	
...	
ExtendedInfo	ServiceVersion

SourceID (8 bytes): The identifier of the system that published the activation message. The value of the **SourceID** field MUST be identical to the value of the **ActivationChannelID** field that the sending system also published within its **Service Descriptor** message (section 2.2.8). This identifier is not used as the reply ChannelID for this message; it is used for debugging and role determination. If the activation message requires a reply message, the reply ChannelID MUST be specified in the body of the specific activation message rather than in this header.

ServiceActivationUUID (16 bytes): A UUID that specifies the service being activated.

ExtendedInfo (2 bytes): This field is primarily provided for 32-bit alignment of the **Service Activation** header. All service protocols defined in this specification require that this field SHOULD be zero; however, it can safely be ignored by receivers. Other service protocols might require other uses for this field.

ServiceVersion (2 bytes): An unsigned integer that specifies the version of the service being activated. The value MUST be nonzero; service activations containing a zero service version MUST be ignored. The first version of all service protocols MUST be 1. A peer that supports version *X* of a given service MUST support activations with versions 1 through *X*.

2.2.8 Service Descriptor Message

The **Service Descriptor** message MUST be published and subscribed at the following well-known channel: "Windows.windows.com/SD".

The length of the **Service Descriptor** message MUST be provided by the transport layer in order to allow the determination of the number of **Service Descriptor (SD)** structures (section 2.2.9) contained within it. Each **SD** structure MUST be fully decoded before being accepted, and if a partial structure occurs at the end of the **Service Descriptor** message, it MUST be ignored.

The **Service Descriptor** message format is specified as follows.

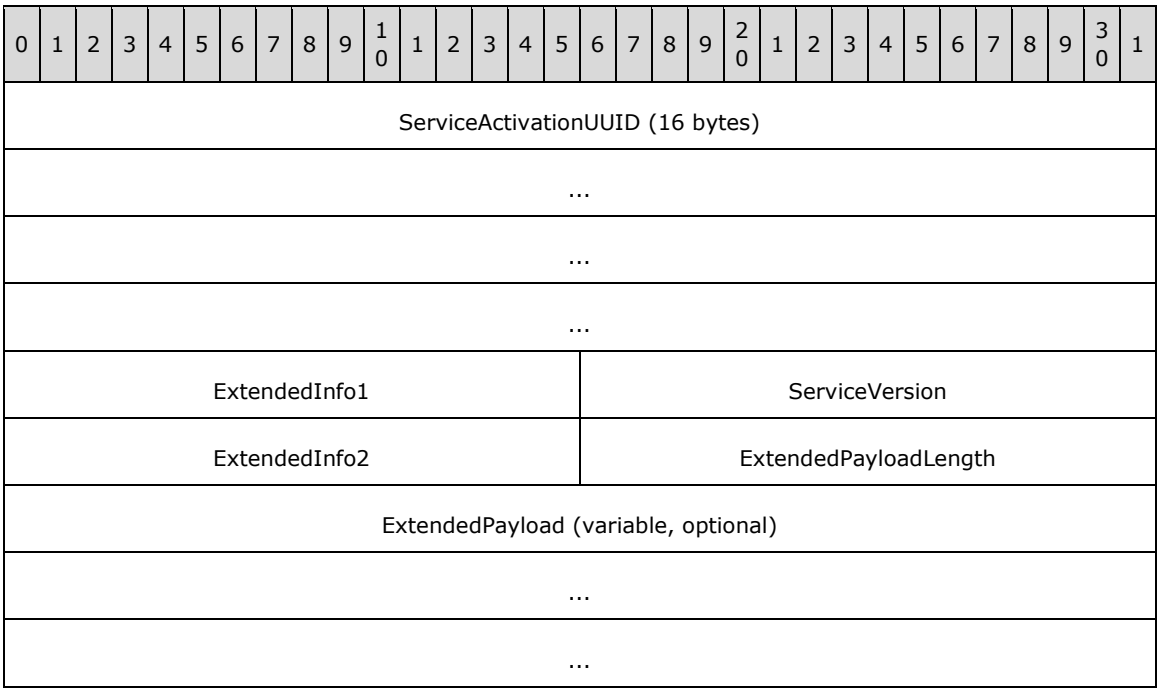
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ActivationChannelID																															
...																															
ServiceDescriptorArray (variable)																															
...																															
...																															

ActivationChannelID (8 bytes): The source identifier of the system that published the **Service Descriptor** message. This value SHOULD be used as the reply ChannelID by the receiver for any activation messages.

ServiceDescriptorArray (variable): Some number of **SD** structures. The **SD** structure format is specified in section 2.2.9.

2.2.9 Service Descriptor Structure

The **Service Descriptor (SD)** structure specifies a service to be activated and provides for explicit capability negotiation. An array of **SD** structures is specified in a **Service Descriptor** message (section 2.2.8). The **SD** structure format is specified as follows.



ServiceActivationUUID (16 bytes): The UUID of the specific service being activated.

ExtendedInfo1 (2 bytes): Service-specific extension information. Each service protocol SHOULD define what this field is used for**.**

ServiceVersion (2 bytes): A positive integer that specifies the version of the service being activated. The value SHOULD be nonzero; service activations containing a zero service version MUST be ignored. The first version of all service protocols MUST be 1. A peer that claims to support version *X* of a given service MUST support activations with versions 1 through *X*.

ExtendedInfo2 (2 bytes): Service-specific extension information. Each service protocol SHOULD define what this field is used for**.**

ExtendedPayloadLength (2 bytes): The length in bytes of additional service-specific extension information in the **ExtendedPayload** field. Each service protocol SHOULD define whether or not the Extended Payload is used. If this field is nonzero, but there are not enough bytes left in this message, then this last entry is ill-formed and MUST be ignored.

ExtendedPayload (variable, optional): Additional service-specific extension information. Each service protocol defines whether or not this information is used, and if so, how it is used.

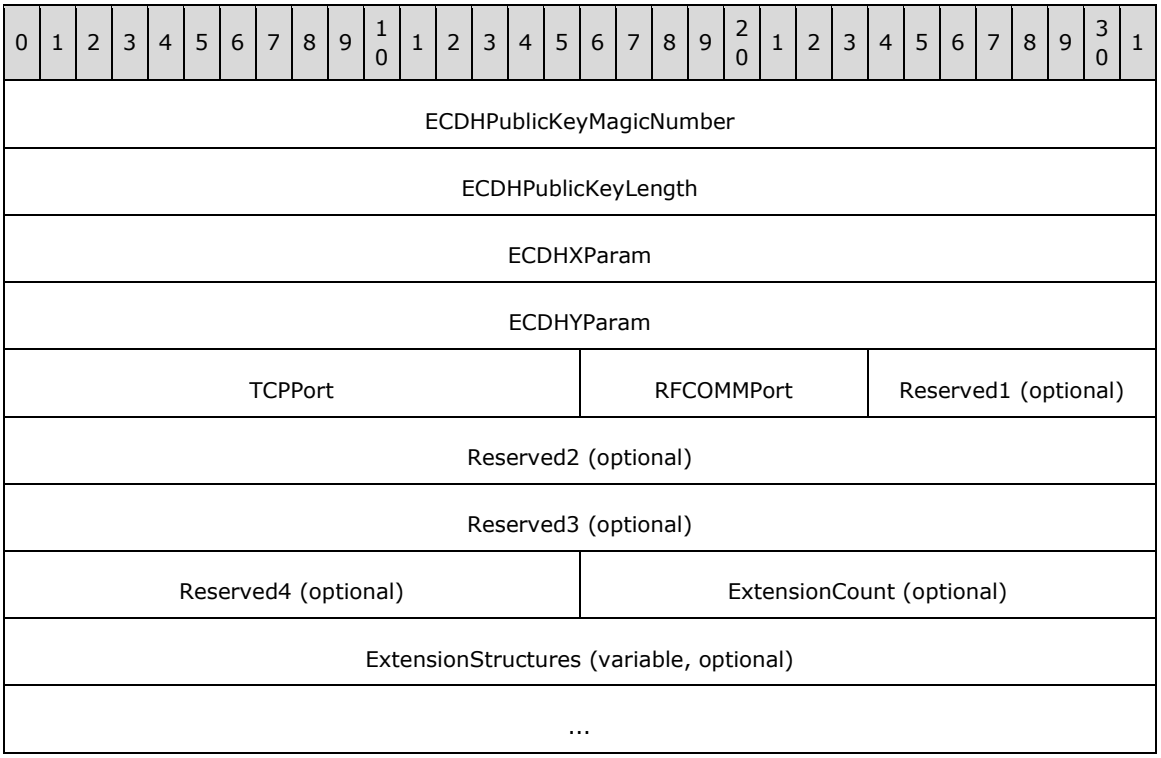
The service listed in the **SD** MUST have a positive integer **ServiceVersion** value associated with it. If an implementation specifies a service with version *X* in its published **SD**, the implementation MUST be compatible with all version numbers less than or equal to *X*. If an implementation cannot support prior versions, it MUST specify a new **ServiceActivationUUID**, in effect creating a new

service. A **Service Activation** header (section 2.2.7) contains the UUID of the service to activate and the service version number.

2.2.10 Session ACK Message

The **Session ACK** message is the acknowledgment/reply to the Session Activation message (section 2.2.11). The transport-provided length of this message MUST be used by the receiver in order to determine the existence or number of extension structures that follow the non-optional portions of the message. The publisher MUST NOT publish messages that are less than 75 bytes long to a Session Activation message's ReplyChannelID. The subscriber MUST drop all Session ACK messages that are less than 75 bytes long.

The **Session ACK** message format is specified as follows.



ECDHPublicKeyMagicNumber (4 bytes): The 4-byte value 0x45, 0x43, 0x4B, and 0x31. This indicates that the Elliptic Curve Diffie-Hellman (ECDH) key exchange follows the P256 convention [NSA].

ECDHPublicKeyLength (4 bytes): A 32-bit, unsigned integer in little-endian format that specifies the key length in bytes. This value MUST be 0x00000020.

ECDHXParam (4 bytes): A 32-bit, unsigned integer that specifies the X coordinate of a single-use, generated ECDH public key. The private key portion MUST NOT be transmitted and is used in the local **Session** object (section 3.1.1.4) with the ECDH Public Key received from the **Session Activation** message (section 2.2.11).

ECDHYParam (4 bytes): A 32-bit, unsigned integer that specifies the Y coordinate of a single-use, generated ECDH public key. The private key portion MUST NOT be transmitted and is used in the local **Session** object with the ECDH Public Key received from the **Session Activation** message.

TCPPort (2 bytes): The TCP port that the publisher's session is listening on.

RFCOMMPort (1 byte): The RFCOMM port that the publisher's session is listening on.

Reserved1 (1 byte, optional): If present, this field MUST be set to zero when sent and MUST be ignored when received.

Reserved2 (4 bytes, optional): If present, this field MUST be set to zero when sent and MUST be ignored when received.

Reserved3 (4 bytes, optional): If present, this field MUST be set to zero when sent and MUST be ignored when received.

Reserved4 (2 bytes, optional): If present, this field MUST be set to zero when sent and MUST be ignored when received.

ExtensionCount (2 bytes, optional): The number of **Extension** structures (section 2.2.3) in the **ExtensionStructures** field. Use of this field is platform specific. A subscriber MUST ignore extensions that it does not process.

If this message is between 75 and 87 bytes long (inclusive), the value of this field is treated as zero.

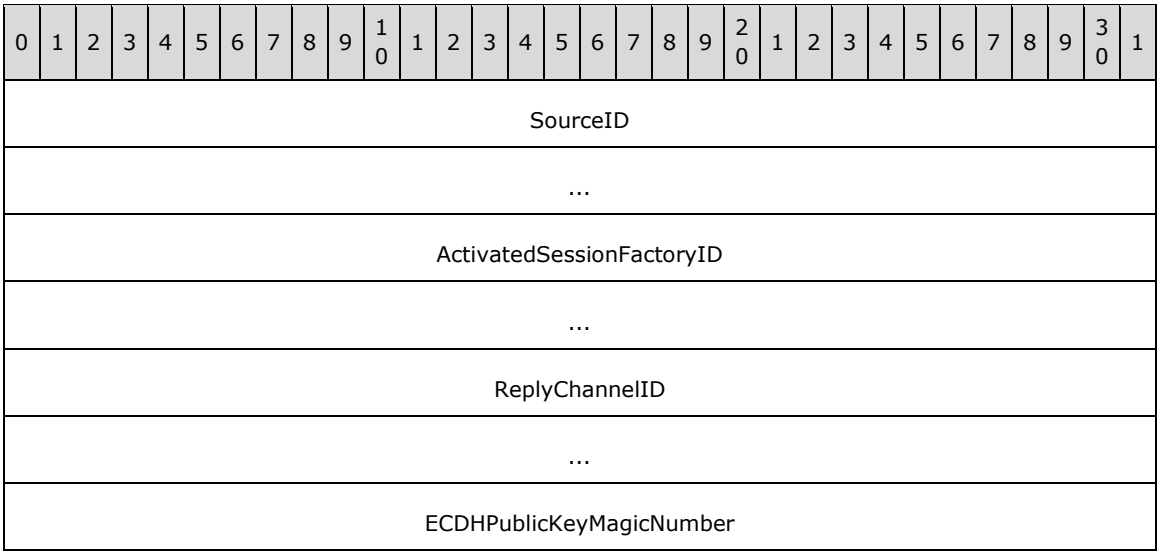
ExtensionStructures (variable, optional): Zero or more **Extension** structures. Any incorrectly formatted **Extension** structures MUST be ignored by the subscriber.

2.2.11 Session Activation Message

The **Session Activation** message is a reply to the **Session Factory Service Activation** message (section 2.2.12). It is used to establish a paired set of **Session** objects (section 3.1.1.4) between two peers. For each active **Session Factory** object (section 3.1.1.3) on the receiver, a **Session Activation** message MUST be published on the **Session Factory Service Activation** message's **ReplyChannelID**, unless the **Session** object is already created and active.

The transport-provided length of this message MUST be used by the receiver in order to determine the existence or number of extension structures that follow the non-optional portions of the message. The publisher MUST NOT publish messages to a **Session Factory Service Activation** message's **ReplyChannelID** that are less than 96 bytes long. The subscriber MUST drop all **Session Activation** messages that are less than 96 bytes long.

The **Session Activation** message format is specified as follows.



ECDHPublicKeyLength	
ECDHXParam	
ECDHYParam	
Reserved1	
Reserved2	
Reserved3	ExtensionCount
ExtensionStructures (variable)	
...	
...	

SourceID (8 bytes): The publisher identification to the subscriber. The publisher of this message MUST set this field to the value of the **ActivationChannelID** field specified in the **Service Descriptor** message that is the source identifier of the publisher. This allows a created **Session** object for the peers to reference an **OOB Connector** object (section 3.1.1.2) for the same two peers.

ActivatedSessionFactoryID (8 bytes): The identifier of the active **Session Factory** object that was activated by the **Session Factory Service Activation** message.

ReplyChannelID (8 bytes): The identifier of the newly created **Session** object. It MUST be generated at random by the publisher. The publisher of this message MUST subscribe to the **ReplyChannelID** prior to publishing this message to ensure that replies are not missed. The publisher MUST handle **Session ACK** messages (section 2.2.10) on this channel.

ECDHPublicKeyMagicNumber (4 bytes): The 4-byte value 0x45, 0x43, 0x4B, and 0x31. This indicates that the Elliptic Curve Diffie-Hellman (ECDH) key exchange follows the P256 convention [NSA].

ECDHPublicKeyLength (4 bytes): A 32-bit, unsigned integer in little-endian format that specifies the key length in bytes. This value MUST be 0x00000020.

ECDHXParam (4 bytes): A 32-bit, unsigned integer that specifies the X coordinate of a single-use, generated ECDH public key. The private key portion MUST NOT be transmitted and is held in the local **Session** object for later use when the **Session ACK** message is received.

ECDHYParam (4 bytes): A 32-bit, unsigned integer that specifies the Y coordinate of a single-use, generated ECDH public key. The private key portion MUST NOT be transmitted and is held in the local **Session** object for later use when the **Session ACK** message is received.

Reserved1 (4 bytes): This field MUST be set to zero when sent and MUST be ignored when received.

Reserved2 (4 bytes): This field MUST be set to zero when sent and MUST be ignored when received.

Reserved3 (2 bytes): This field MUST be set to zero when sent and MUST be ignored when received.

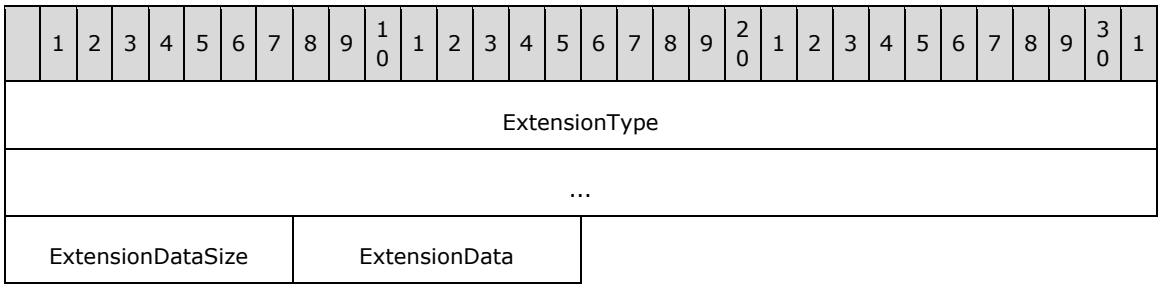
ExtensionCount (2 bytes): The number of **Extension** structures (section 2.2.3) in the **ExtensionStructures** field. Use of this field is platform-specific. A subscriber MUST ignore extensions that it does not process.

If this message is between 96 and 107 bytes long (inclusive), the value of this field is treated as zero.

ExtensionStructures (variable, optional): Zero or more **Extension** structures. Any incorrectly formatted **Extension** structures MUST be ignored by the subscriber.

If the **Session Factory Service Activation** message to which this **Session Activation** message is a response contains a **Role** field, the following additional requirements for this message are defined:<2>

- The input **Role** field value SHOULD be checked, as specified in section 3.1.5.6.
- The **Extension** structure in the **ExtensionStructures** field SHOULD be structured as follows.



ExtensionType (8 bytes): The value 0x89A14CC3AB4CF821.

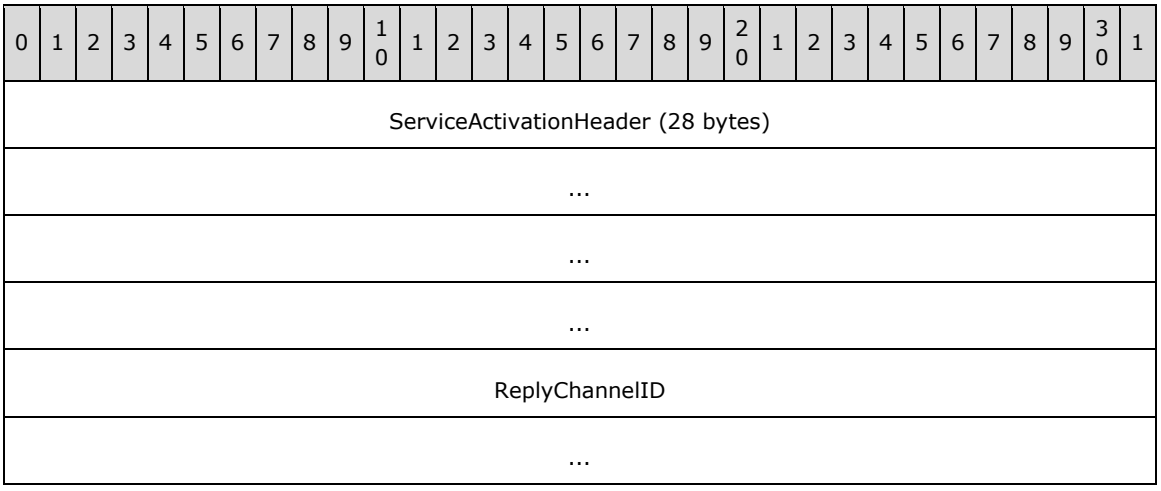
ExtensionDataSize (1 byte): The value 0x01.

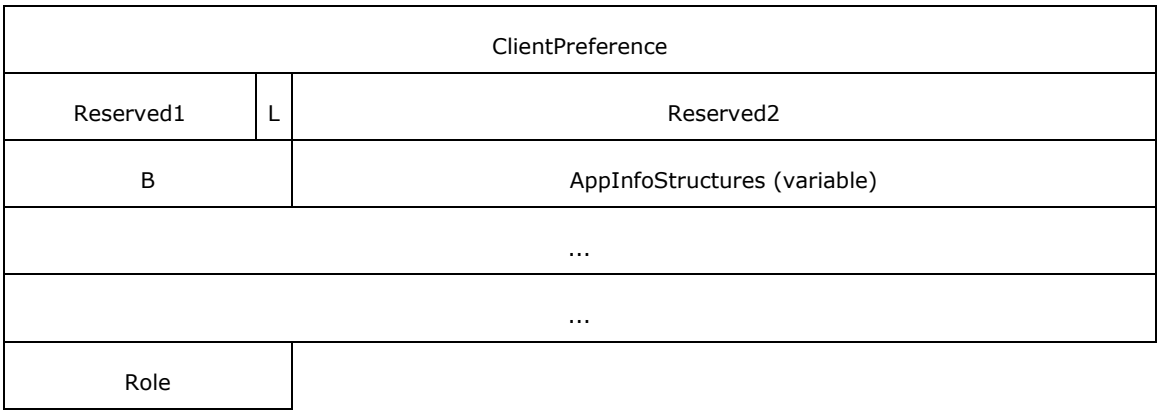
ExtensionData (1 byte): The compatible role value according to the **Role Compatibility** constants (section 2.2.6).

2.2.12 Session Factory Service Activation Message

The **Session Factory Service Activation** message is a reply to the **Service Descriptor** message (section 2.2.8). It is used to establish a paired set of **Session** objects (section 3.1.1.4) between two peers. For each active **Session Factory** object (section 3.1.1.3) on the receiver, a **Session Activation** message (section 2.2.11) MUST be published on the **ReplyChannelID** of the **Session Factory Service Activation** message, unless the **Session** object is already created and active.

The **Session Factory Service Activation** message format is specified as follows.





ServiceActivationHeader (28 bytes): A **Service Activation** header (section 2.2.7). The **ServiceActivationUUID** value SHOULD<3> be {F1DEBC56-CFBA-4129-983B-7D79499D1A7D} for a peer role or {DAA42D35-1323-485A-8B34-3B86E416E6EC} for a host or client role. If it is the latter, the **Role** field MUST be included in this message, as specified later in this section.

The **ServiceVersion** value in the **Service Activation** header MUST be 1.

ReplyChannelID (8 bytes): This value is also the identifier of the **Session Factory** object (section 3.1.1.3). It MUST be generated at random by the publisher. The publisher of this message MUST subscribe to the **ReplyChannelID** prior to publishing this message to ensure that replies are not missed. The publisher MUST handle **Session Activation** messages on this channel.

ClientPreference (4 bytes): The preference of the sender to be the peer that actually sends the subsequent **Session Activation** message. Values higher than 0x1000 indicate a preference to have **Session** objects be the client role. Values lower than 0x1000 indicate a preference to have **Session** objects be the server role.

Reserved1 (7 bits): This field MUST be set to zero by publishers and MUST be ignored by subscribers.

L (1 bit): The Launch flag. The presence of this flag indicates the publisher's intent for the application on the subscriber side to be launched or activated if it is not already. If this flag is not set, the subscriber MUST NOT trigger the launching of the application specified in the **AppInfo** structure (section 2.2.2).

Reserved2 (3 bytes): This field MUST be set to zero by publishers and MUST be ignored by subscribers.

AppInfoCount (1 byte): The number of **AppInfo** structures that follow this field. The publisher MUST provide 1 or more **AppInfo** structures. If this field is zero, the entire message MUST be ignored by the subscriber.

AppInfoStructures (variable): An array of **AppInfo** structures. Typically the first one uniquely refers to the application on the platform that published this message. The publisher can provide application information that refers to the application on other platforms for the purposes of interoperation.

Role (1 byte, optional): The session role of the application. This field SHOULD<4> be included in this message if the application role is either host or client. The following values are valid.

Value	Description
0x02	Host role
0x03	Client role

3 Protocol Details

3.1 Peer Details

This section defines peer roles in the Near Field Proximity: Bidirectional Services Protocol.

In a socket-based connection between two peer applications, one peer has the role of client, and the other peer has the role of server. The roles are distinguished as follows:

- The client is the peer that sends the **Session Activation** message (section 2.2.11) and waits for the **Session ACK** message (section 2.2.10).
- The server is the peer that receives the **Session Activation** message and sends the **Session ACK** message.

Possible states and state transitions of the client and server roles are described in section 3.1.1.4.

In an OOB connection between two peers, one peer has the role of connector, and the other peer has the role of listener. The roles are distinguished as follows:

- The connector is the peer that sends the **OOB Connector Service Activation** message (section 2.2.5) and waits for the **OOB Connector Service ACK** message (section 2.2.4).
- The listener is the peer that receives the **OOB Connector Service Activation** message and sends the **OOB Connector Service ACK** message.

Possible states and state transitions of the connector and listener roles are described in section 3.1.1.2.

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.

The abstract data model defines **OOB Connector** objects, **Session Factory** objects, and **Session** objects. When the underlying transport is triggered, an exchange is performed by peers that can result in new instances of these objects: each peer can create an **OOB Connector** object, one in the listener role and one in the connector role. If both sides have active **Session Factory** objects that are compatible, then each peer creates a **Session** object, one in the client role and one in the server role.

Each **Session** object references one **OOB Connector** object in order to provide connectivity options to higher-level protocols.

If there is only one peer with an active **Session Factory** object, but it specifies the **L** (Launch) flag in the **Session Factory Service Activation** message (section 2.2.12), then the other peer can create a **Session Factory** object on behalf of a soon-to-be-launched application. This provides for the ability to establish OOB connections at the same time as launching an application.

Note that the abstract interface notation (Public) indicates that the abstract data model element can be directly read or written from outside this protocol by higher-level protocols. For example, the Near Field Proximity: Sharing Protocol [MS-NFPS] uses members of the **Session** and **OOB Connector** objects to construct a socket for the purposes of sharing file(s).

3.1.1.1 NfpService

The Near Field Proximity service **NfpService** encapsulates the entire state for the protocols described by this document.

SourceID (8 bytes): A random number that uniquely identifies the **NfpService** instance.

OOBConnectorList: A list of active **OOB Connector** objects.

SessionFactoryList (Public): A list of active **Session Factory** objects.

HandshakeData: The **SessionID** and **ConnectionType** handshake data from the **Accept Header** (section 2.2.1) that was used to confirm the connection.

3.1.1.2 OOB Connector Object

An **OOB Connector** object encapsulates the state for an OOB connection between two peers.

Role: The role of the **OOB Connector** object. One peer is the connector, and the other peer is the listener. The roles are distinguished as follows:

- The connector is the peer that sends the **OOB Connector Service Activation** message (section 2.2.5) and waits for the **OOB Connector Service ACK** message (section 2.2.4).
- The listener is the peer that receives the **OOB Connector Service Activation** message and sends the **OOB Connector Service ACK** message.

State: The current state of the **OOB Connector** object. The meaning of the state depends on the object role; this allows an asymmetric client/server message exchange. For the connector role, the state can be one of the following.

Value	Meaning
WaitingForAck	The object has published the **OOB Connector Service Activation** message (section 2.2.5) and is waiting to receive the **OOB Connector Service ACK** message (section 2.2.4).
Incomplete	The OOB Connector protocol timed out. The object is still alive to facilitate object continuity across multiple taps.
Ready	The object has received the **OOB Connector Service ACK** message and has all the information required to facilitate communications between the two peers.

The following is a diagram that shows the state transitions for the connector role of an **OOB Connector** object.

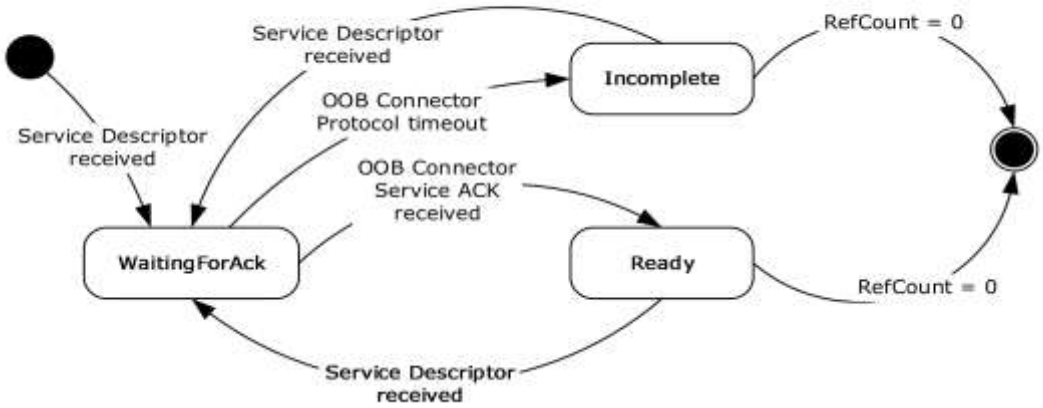


Figure 4: OOB Connector state transitions: Connector role

For the listener role, the state can be one of the following.

Value	Meaning
WaitingForTransmit	The object has received the **OOB Connector Service Activation** message, has published the **OOB Connector Service ACK** message to the transport, and is waiting for the transport to indicate the ACK message has been transmitted to a remote peer
Incomplete	The OOB Connector protocol timed out. The object is still alive to facilitate object continuity across multiple taps.
Ready	This object has received notification of successful transmission of the ACK message and has all the information required to facilitate communication between two peers.

The following is a diagram that shows the state transitions for the listener role of an **OOB Connector** object.

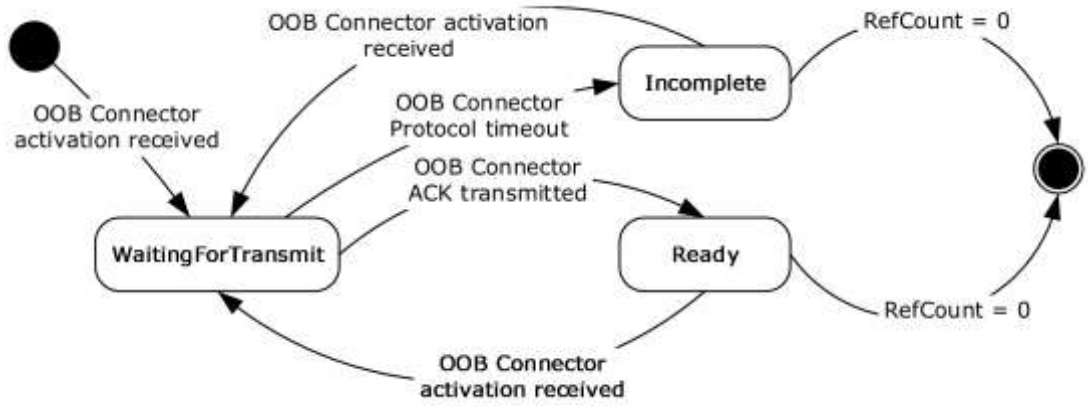


Figure 5: OOB Connector state transitions: Listener role

RemoteSourceID: This identifies the peer to which a given **OOB Connector** object is connected. For the connector, this is copied from the **ActivationChannelID** field in the received **Service Descriptor** message (section 2.2.8). For the listener, this is copied from the **SourceID** field in

the **Service Activation** header (section 2.2.7) of the received **OOB Connector Service Activation** message.

OOBConnectorID: For the connector, this is randomly generated and used in the **OOB Connector Service Activation** message. For the listener, this is copied from the received **OOB Connector Service Activation** message.

WFDPeerConnected: A Boolean value that indicates whether the **OOB Connector** object has an active Wi-Fi Direct (WFD) connection to the peer.

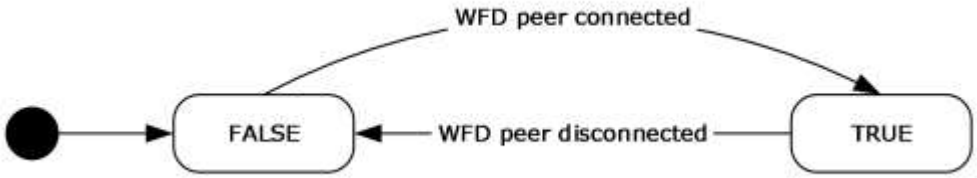


Figure 6: OOB Connector WFDPeerConnected transitions

LocalAddresses (Public Read): The list of local addresses collected by the **OOB Connector** object from the local machine and sent to the remote machine via either the Activation or ACK message.

RemoteAddresses (Public Read): The list of remote addresses received by the **OOB Connector** object from the remote machine via either the Activation or ACK message.

ReferenceCount: A count of references to the **OOB Connector** object by either **Session** objects or KeepAlive timers.

3.1.1.3 Session Factory Object

A **Session Factory** object encapsulates a factory for socket-based connections between set(s) of peer applications.

SessionList: A list of **Session** objects (section 3.1.1.4) that are a part of this **Session Factory** object.

AppID (Public Read/Write): A UTF-8 string that identifies the contract or interface for this **Session Factory** object. This **Session Factory** will only create and link up **Session** objects with other **Session Factory** objects that provide exactly the same **AppID** in the **AppInfo** field of the **Session Factory Service Activation** message (section 2.2.12).

AlternateIDList (Public Read/Write): A list of alternate **AppIDs** for other platforms that the **Session Factory** will also attempt to activate.

Launch (Boolean): TRUE if the **L** (Launch) flag SHOULD be set in the appropriate **Session Factory Service Activation** message. FALSE if the flag SHOULD be cleared.

SessionFactoryID (8 bytes): A random number that uniquely identifies this instance of **Session Factory** object.

TcpPort (Public Read): The TCP/IP port on which the **Session Factory** is listening and can accept **Session** sockets after a **Session Activation** (section 2.2.11)/ **Session ACK** (section 2.2.10) exchange.

RfcommPort (Public Read): The RFCOMM/Bluetooth port on which the **Session Factory** is listening and can accept **Session** sockets after a **Session Activation**/**Session ACK** exchange.

ReferenceCount (Public Write): A count of references to the **Session Factory** object by either the client application or KeepAlive timers.

3.1.1.4 Session Object

A **Session** object encapsulates the state for a socket-based connection between two peer applications.

Role (Public Read): The role of the **Session** object. One peer is the client, and the other peer is the server. The roles are distinguished as follows:

- The client is the peer that sends the **Session Activation** message (section 2.2.11) and waits for the **Session ACK** message (section 2.2.10).
- The server is the peer that receives the **Session Activation** message and sends the **Session ACK** message.

State (Public Read/Write): The current state of the **Session** object. The state can be one of the following.

Value	Meaning
WaitingForAck	A client **Session** object transitions to this state immediately prior to publishing the **Session Activation** message.
WaitingForTransmit	A server **Session** object transitions to this state when beginning to publish the **Session ACK** message.
Ready	The **Session** object is ready to be used by an application for peer-to-peer communication. A client **Session** object transitions to this state after receiving the **Session ACK** message. A server **Session** object transitions to this state after successfully transmitting the **Session ACK** message.
Terminated	The **Session** object has been terminated by the application, or it timed out.

The following shows the possible state transitions for the client role of a **Session** object.

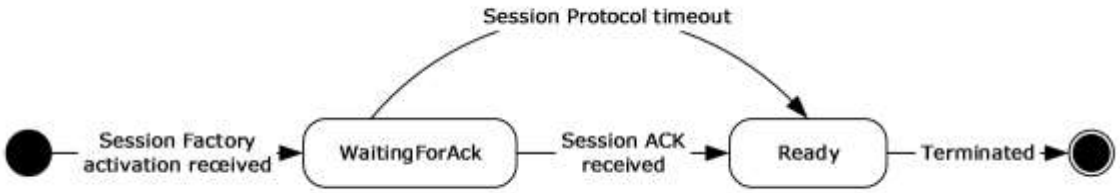


Figure 7: Session state transitions: Client role

The following shows the possible state transitions for the server role of a **Session** object.

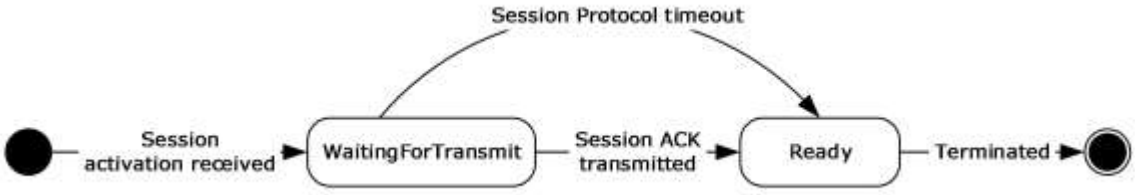


Figure 8: Session state transitions: Server role

SessionID: For the client, this is randomly generated and used in the **Session Activation** message. For the server, this is copied from the **ReplyChannelID** field of the received **Session Activation** message. The server uses this ID to publish the **Session ACK** message.

RemoteSessionFactoryID (8 bytes): The SessionFactoryID of the remote peer this **Session** object is connected to. For the client, this is copied from the **ReplyChannelID** field of the received **Session Factory Service Activation** message (section 2.2.12). For the server, this is copied from the **ActivatedSessionFactoryID** field of the received **Session Activation** message.

RemoteTcpPort (Public Read): The TCP/IP port on which the remote **Session Factory** is listening and can accept **Session** sockets after a **Session Activation**/**Session ACK** exchange.

RemoteRfcommPort (Public Read): The RFCOMM/Bluetooth port on which the remote **Session Factory** is listening and can accept **Session** sockets after a **Session Activation**/**Session ACK** exchange.

PrivateKey: The private key used in an Elliptic Curve Diffie-Hellman (ECDH) exchange to derive a shared key. The client and server have different private keys and do not share them with anyone.

PublicKey: The public key (linked with the **PrivateKey**) used in an Elliptic Curve Diffie-Hellman (ECDH) exchange to derive a shared key. The client and the server have different public keys and exchange them in the **Session Activation** and **Session ACK** messages.

SharedSecretKey (Public Read): The key derived from the ECDH key exchange. This key is not published and SHOULD remain a shared secret between the server and clients. The key can be provided to the application on each side so that it can provide a level of authentication/encryption over the Session link.

ReferencedOOBConnector: A link to the **OOB Connector** object that holds OOB connection information for the two peers linked by the **Session** object. When present, this represents a reference count on the **OOB Connector** object.

3.1.2 Timers

The following timers are used by this protocol.

SessionProtocolTimer: An independent timer for each **Session** object (section 3.1.1.4), which causes the **Session** object to time out if its **State** is not Ready.

The legal range for the **SessionProtocolTimer** timeout value is 8-60 seconds.<5>

OOBConnectorProtocolTimer: An independent timer for each **OOB Connector** object, which is started each time an **OOB Connector** object **State** transitions into WaitingForTransmit or WaitingForACK. If the timer fires before the object **State** transitions to Ready, the protocol is incomplete and the object **State** transitions to Incomplete.

The legal range for the **OOBConnectorProtocolTimer** timeout value is 8-60 seconds.<6>

The following timers illustrate abstract data model object lifetime issues, but they are not necessary in all implementations:

OOBConnectorKeepAliveTimer: An independent timer for each **OOB Connector** object (section 3.1.1.2), which is started each time an **OOB Connector** object **State** transitions into Incomplete. If it fires before the object **State** transitions out of Incomplete, the initialization reference SHOULD be released.<7> If there are no other references to the object, the object is destroyed.

SessionFactoryKeepAliveTimer: An independent timer for each **Session Factory** object (section 3.1.1.3), which is started when a **Session Factory** object is created. If it fires before a client takes ownership of the object, the **Session Factory** object SHOULD BE deleted.<8>

3.1.3 Initialization

The **NfpService** MUST be initialized prior to being useful to any higher-level protocol; initializing at system startup is sufficient. On initialization:

- A subscription MUST be made on a **Service Descriptor** message (section 2.2.8) well-known channel.
- A **SourceID** MUST be randomly generated.
- A subscription for **Session Activation** messages (section 2.2.11) MUST be made by using the **SourceID** as the ChannelID.
- A local **Service Descriptor** message MUST be constructed, which will be published as required, based on higher-layer triggered events:
  - The message MUST use the **SourceID** of the **NfpService**.
  - The message MUST contain one **Service Descriptor (SD)** structure (section 2.2.9) for the **OOB Connector** service with the following:
    - **ServiceActivationUUID**: {E46EDA50-9B5D-41F1-B89E-327B5EA38B16}
    - **ServiceVersion**: 1
    - **ExtendedInfo1**: Zeros
    - **ExtendedInfo2**: Zeros
    - **ExtendedPayloadLength**: Zero
- The **Service Descriptor** message MUST contain one **SD** structure for **Session Factory Service Activation** with the following values:
  - **ServiceActivationUUID**: {F1DEBC56-CFBA-4129-983B-7D79499D1A7D}
  - **ServiceVersion**: 1
  - **ExtendedInfo1**: Zeros
  - **ExtendedInfo2**: Zeros
  - **ExtendedPayloadLength**: Zero

3.1.4 Higher-Layer Triggered Events

Higher-layer protocols use this protocol by creating **Session Factory** objects. When an active **Session Factory** object is created, the NfpService MUST ensure that its local **Service Descriptor** (section 2.2.9) is published to the transport. Higher-layer protocols like the Near Field Proximity: Sharing Protocol [MS-NFPS] can also register to handle launching of applications that can use the created socket connection. The implementation for this is not specified.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Service Descriptor Sequence

The following list defines the required actions of the **NfpService** if an incoming **Service Descriptor** message (section 2.2.8) is received on a transport link:

- The local **Service Descriptor (SD)** structure (section 2.2.9) MUST be published, unless it has already been sent on the current active transport link.
- The local **SD** MUST NOT be published twice on any one active transport link.
- If the incoming message contains an **SD** structure with an **OOB Connector** service UUID as specified in section 2.2.9, the **OOB Connector** exchange MUST be followed as specified in section 3.1.5.2.
- If the incoming message contains an **SD** structure with a **Session Factory** service UUID and an **OOB Connector** service UUID as specified in section 2.2.9, the **Session Factory** exchange MUST be followed as specified in section 3.1.5.5.

3.1.5.2 OOB Connector Exchange

The following sequence defines the required actions of the **NfpService** if a **Service Descriptor** message (section 2.2.8) received on the transport link contains an **SD** structure with a valid **OOB Connector** service UUID as specified in section 2.2.9:

1. If the received **ActivationChannelID** is equal to the **SourceID** of the local **NfpService**, then this sequence MUST be stopped: Steps 2 and later MUST NOT occur.
2. If the received **ActivationChannelID** is greater than or equal to the **SourceID** of the local **NfpService**, then this sequence MUST be stopped: Steps 3 and later MUST NOT occur.

   Note The remote peer is the one that continues the **OOB Connector** exchange.
3. A new **OOB Connector** object SHOULD be created in the **NfpService** list with the following attributes, unless one already exists with a **RemoteSourceID** equal to the **ActivationChannelID** field of the received **Service Descriptor** message:
   1. **Role**: Connector
   2. **RemoteSourceID**: The received **ActivationChannelID**
   3. **OOBConnectorID**: This ID is randomly generated.
   4. **WFDPeerConnected**: FALSE
4. If the **OOB Connector** object has the listener role, then this sequence MUST be stopped: Steps 5 and later MUST NOT occur.
5. Reset the following **OOB Connector** object variables:
   1. **State**: WaitingForAck
   2. **LocalAddresses**: The list of local addresses collected from the local machine.
6. If the **OOB Connector** object's **WFDPeerConnected** field is equal to FALSE, construct a new WFD connect BLOB for the local machine.
7. Subscribe to the **OOB Connector Service ACK** message (section 2.2.4) on the **OOB Connector** object's **OOBConnectorID**.
8. Construct an **OOB Connector Service Activation** message (section 2.2.5):
   1. Set the **ServiceActivationHeader**, **WiFiDirectConnectBlobLength**, and **WiFiDirectConnectBlob** fields.
   2. Use the **OOB Connector** object's **OOBConnectorID** element to set the **ReplyChannelID** field.

3. Use the **OOB Connector** object's **LocalAddresses** element to set the various **Address** fields.

9. Publish the **OOB Connector Service Activation** message on the received **ActivationChannelID**.

10. If the **OOBConnectorProtocolTimer** (section 3.1.2) has not yet fired for this **OOB Connector** object, stop the timer and decrement the **OOB Connector** object's **ReferenceCount** element by 1.

11. Start the **OOBConnectorProtocolTimer** for this **OOB Connector** object.

12. Increment the **OOB Connector** object's **ReferenceCount** element by 1.

3.1.5.3 Handling OOB Connector Service Activation Messages

The following sequence defines the required actions of the **NfpService** if an **OOB Connector Service Activation** message (section 2.2.5) is received on the transport link on the **SourceID** of the **NfpService**.

1. A new **OOB Connector** object SHOULD be created in the NfpService's list with the following attributes, unless one already exists with a **RemoteSourceID** equal to the **SourceID** field in the **Service Activation** header (section 2.2.7) within the received message:

    1. **Role**: Listener

    2. **RemoteSourceID**: The **SourceID** field in the **Service Activation** header within the received message.

    3. **WFDPeerConnected**: FALSE

2. If the **OOB Connector** object has the connector role, then this sequence MUST be stopped: Steps 5 and later MUST NOT occur.

3. Reset the following **OOB Connector** object variables:

    1. **State**: WaitingForTransmit

    2. **OOBConnectorID**: Copy this value from the **ReplyChannelID** field of the received message.

    3. **LocalAddresses**: The list of local addresses collected from the local machine.

    4. **RemoteAddresses**: Copy this value from the various **Address** fields of the received message.

4. If the **OOB Connector** object's **WFDPeerConnected** field is equal to FALSE, construct a new WFD listen BLOB for the local machine.

5. If the **OOB Connector** object's **WFDPeerConnected** field is equal to FALSE, the **NfpService** can attempt to use the WFD connect BLOB to start listening for WFD OOB pairing connections.

6. Construct an **OOB Connector Service ACK** message (section 2.2.4):

    1. Set the **WiFiDirectListenBlobLength** and **WiFiDirectListenBlob** fields as specified in section 2.2.4.

    2. Use the **OOB Connector** object's **LocalAddresses** to set the various address fields.

7. Publish the **OOB Connector Service ACK** message on the received **ReplyChannelID**.

If the transport link indicates that this message is transmitted prior to the **OOBConnectorProtocolTimer** expiring for this **OOB Connector** object, then the **OOB Connector** object's **State** moves to Ready.

1. If the **OOBConnectorProtocolTimer** has not yet fired for this **OOB Connector** object, stop the timer and decrement the **OOB Connector ReferenceCount** by 1.
2. Start the **OOBConnectorProtocolTimer** for this **OOB Connector** object.
3. Increment the **OOB Connector ReferenceCount** by 1.

3.1.5.4 Handling OOB Connector Service ACK Messages

The following sequence defines the required actions of the **NfpService** if an **OOB Connector Service ACK** message (section 2.2.4) is received on the transport link on the **OOBConnectorID** ChannelID of a specific **OOB Connector** object:

1. If the **OOB Connector** object's **State** is NOT WaitingForAck, then this sequence MUST be stopped: Steps 2 and later MUST NOT occur.
2. Reset the following **OOB Connector** object variables:
    1. **State**: Ready
    2. **RemoteAddresses**: Copy this value from the various **Address** fields of the received message.
3. If the **OOB Connector** object's **WFDPeerConnected** field is equal to FALSE, the **NfpService** can attempt to use the received WFD listen BLOB and the previously sent WFD connector BLOB to initiate a WFD OOB pairing connection.

3.1.5.5 Session Factory Exchange

The following sequence defines the required actions of the **NfpService** if a **Service Descriptor** message (section 2.2.8) is received on the transport link with both a valid **OOB Connector** service UUID and a valid **Session Factory** service UUID. Both MUST be present because the **Session Factory** service relies on the **OOB Connector** exchange.

For each **Session Factory** object (section 3.1.1.3) in the **NfpService SessionFactoryList** where the context of the system indicates that the user intends to use the **Session Factory** to link with the Peer, the following can be performed:

1. Construct a **Session Factory Service Activation** message (section 2.2.12):
    1. Set the **Service Activation** header (section 2.2.7), **ClientPreference**, and **Reserved** fields.
    2. Use the **Session Factory** object's **SessionFactoryID** to set the **ReplyChannelID** field in the message.
    3. If the context of the system indicates that the user wants to have the peer launch and/or acquire an application able to handle the Session connection, then the **L** (Launch) flag can be set.
    4. Use the **Session Factory** object's **AppID** and **AlternateIDList** to add one or more **AppInfo** structures (section 2.2.2) to the message.
2. Ensure that a subscription is made on the **ReplyChannelID** (also known as the **SessionFactoryID**) that handles **Session Activation** messages (section 2.2.11).
3. Publish the **Session Factory Service Activation** message on the lower layer transport.

3.1.5.6 Handling Session Factory Service Activation

The following sequence defines the required actions of the **NfpService** if a **Session Factory Service Activation** message (section 2.2.12) is received on the transport link on the **SourceID** of the **NfpService**. For each **Session Factory** object (section 3.1.1.3) in the **NfpService SessionFactoryList**:

1. If the received message does not contain an **AppInfo** structure (section 2.2.2) with the local platform qualifier and the **Session Factory** object's **AppID** field, then this sequence MUST be aborted.

Note Although the **AppID** string is platform-dependent, a binary comparison can be performed.

1. If the received **ClientPreference** field is greater than the local client preference, then this sequence MUST be aborted.

2. If the received **ReplyChannelID** field is greater than the **Session Factory** object's **SessionFactoryID**, then this sequence MUST be aborted.

3. If a **Session** object (section 3.1.1.4) can be found in the **Session Factory** object's **SessionList** with a **State** equal to Ready and a **RemoteSessionFactoryID** equal to the received **ReplyChannelID** field, then this sequence MUST be aborted.

4. If the **Session Factory Service Activation** message that is received contains a **Role** field, its value SHOULD<9> be checked for compatibility according to the **Role Compatibility** constants (section 2.2.6). If the **Role** value is incompatible, then this sequence MUST be aborted.

5. A higher-level protocol can begin at this step.

6. Create a new **Session** object and add it to the **Session Factory** object's **SessionList** with the following attributes:

    1. **Role**: Client

    2. **State**: WaitingForAck

    3. **SessionID**: This is randomly generated.

    4. **RemoteSessionFactoryID**: Set to the **ReplyChannelID** field of the received message.

    5. **PrivateKey/PublicKey**: Generate 256-bit key pair using the Elliptic Curve Diffie-Hellman (ECDH) P256 convention [NSA].

    6. **SharedSecretKey**: Zeroed.

7. Attempt to get a reference link to the **OOB Connector** object indexed by the **SourceID** field in the **Service Activation** header (section 2.2.7) within the received **Session Factory Service Activation** message.

If this is not yet available, it MUST be set if it becomes available prior to the **SessionProtocolTimer** expiration for this **Session** object.

1. Start the **SessionProtocolTimer** for this **Session** object.

2. Construct a **Session Activation** message (section 2.2.11):

    1. Set the **SourceID** and various **Reserved** fields.

    2. Set the **ExtensionCount** and **ExtensionStructures** fields. If the **Session Factory Service Activation** message that is received contains a **Role** field, the **Extension** structure in the **ExtensionStructures** field SHOULD be formatted as specified in section 2.2.11.

3. Use the **Session Factory** object's **SessionFactoryID** to set the **ActivatedSession FactoryID** field.

4. Use the **Session** object's **SessionID** to set the **ReplyChannelID** field.

5. Use the **Session** object's **PublicKey** to set the **ECDHPublicKeyMagicNumber** field.

3. Subscribe to **Session ACK** messages (section 2.2.10) using the **Session** object's **SessionID** as the ChannelID.

4. Publish the **Session Activation** message to the received **ReplyChannelID**.

When the transport indicates that this message has been transmitted, the **Session** object's **State** moves to Ready.

3.1.5.7 Handling Session Activation

The following sequence defines the required actions of the **NfpService** if a **Session Activation** message (section 2.2.11) is received on the transport link on the **ReplyChannelID** of a specific **Session Factory** object (section 3.1.1.3):

1. If a **Session** object (section 3.1.1.4) can be found in the **Session Factory** object's **SessionList** with a **State** equal to Ready and a **RemoteSessionFactoryID** equal to the received **ActivatedSessionFactoryID** field, then this sequence MUST be aborted.

2. Create a new **Session** object and add it to the **Session Factory** object's **SessionList** with the following attributes:

    1. **Role**: Server

    2. **State**: WaitingForTransmit

    3. **SessionID**: Set to the received **ReplyChannelID**

    4. **RemoteSessionFactoryID**: Set to the **ActivatedSessionFactoryID** field of the received message

    5. **PrivateKey/PublicKey**: Generate 256-bit key pair using the Elliptic Curve Diffie-Hellman (ECDH) P256 convention [NSA].

    6. **SharedSecretKey**: Derive using the SHA256 key derivation algorithm with the above **PrivateKey** and **PublicKey** pair and the **ECDH Public Key** field of the received message.

3. Start the **SessionProtocolTimer** for this **Session** object.

4. Construct a **Session ACK** message (section 2.2.10):

    1. Set the **Reserved** and **Extension** fields.

    2. Use the **Session** object's **PublicKey** to set the **ECDH Public Key** field.

    3. Use the **Session Factory** object's **TcpPort** to set the **TCP Port** field.

    4. Use the **Session Factory** object's **RfcommPort** to set the **RFCOMM Port** field.

5. Publish the **Session ACK** message to the received **ReplyChannelID**.

If the transport link indicates that this message is transmitted prior to the **SessionProtocolTimer** expiring for this **Session** object, then the **Session** object's **State** moves to Ready.

The **Session** object can then be used by higher-level protocols to create connections to the peer.

3.1.5.8 Handling Session ACK Messages

The following sequence defines the required actions of the **NfpService** if a **Session ACK** message (section 2.2.10) is received on the transport link on the **SessionID** ChannelID of a specific **Session** object (section 3.1.1.4):

1. If the **Session** object's **State** is not WaitingForAck, then this sequence MUST be aborted.
2. Derive the **SharedSecretKey** using the SHA256 key derivation algorithm (see P256 curve in [NSA]) with the **Session** object's **PrivateKey** and **PublicKey** pair and the **ECDHPublicKeyMagicNumber** field of the received message.
3. The **TCP Port** field of the received message is copied to the **Session** object.
4. The **RFCOMM Port** field of the received message is copied to the **Session** object.
5. The **Session** object's **State** moves to Ready.
6. The **Session** object can be used by higher-level protocols to create connections to the peer.

3.1.5.9 Handling the Accept Header

The following sequence defines the required actions following the activation of the session:

1. The server MUST listen on transports according to information that was exchanged with the client in earlier steps of the protocol.
2. The client MUST attempt to connect to the server on all the transports that the server is listening on.
3. The server chooses the best connection and accepts it.
4. On connection establishment, the client MUST send an **Accept Header** (section 2.2.1) to the server.
5. On receipt of the **Accept Header**, the server MUST validate the **SessionID** from the **Accept Header** by comparing it to the **ReplyChannelID** that it received from the client in the **Session Activation** message (section 2.2.11).
6. If the IDs match, the server MUST send the **Accept Header** back to the client. If the IDs do not match, the server MUST abort the connection.
7. The client MUST validate the **Accept Header** received from the server by comparing it to the **Accept Header** it sent before. If the **Accept Headers** match, the negotiation is complete; otherwise, the client MUST abort the connection.

3.1.6 Timer Events

The following timer events are associated with the timers defined by this protocol (section 3.1.2).

SessionProtocolTimer: If this timer fires when the associated **Session** object (section 3.1.1.4) **State** is WaitingForAck or WaitingForTransmit, the object transitions to the Terminated state. This **Session** object is not usable by higher-level protocols. If this timer fires when the **Session** object **State** is Ready, the timer is ignored.

OOBConnectorProtocolTimer: If this timer fires when the associated **OOB Connector** object (section 3.1.1.2) **State** is WaitingForAck or WaitingForTransmit, the object **State** transitions to Incomplete. If this timer fires when the **OOB Connector** object **State** is any other value, the timer is ignored.

OOBConnectorKeepAliveTimer: If this timer fires before the associated **OOB Connector** object **State** transitions out of Incomplete, the reference logged at initialization SHOULD be released. If there are no other references to the object, the object SHOULD be destroyed.

SessionFactoryKeepAliveTimer: An independent timer for each **Session Factory** object (section 3.1.1.3), which is started when a **Session Factory** object is created. If it fires before a client takes ownership of the object, the **Session Factory** object SHOULD be deleted.

3.1.7 Other Local Events

None.

4 Protocol Examples

The following scenario shows a successful real-time connection established between two peers, Peer A and Peer B. The example demonstrates this through a hypothetical application called Adventure Works, made by Contoso. This example assumes that the underlying transport works like Near Field Communication (NFC), in that the transport is activated when two peers become proximate. Peer A has the **NfpService** (section 3.1.1.1) initialized with an active **Session Factory** object (section 3.1.1.3) that is configured to establish a session when the transport is next activated. Peer A is running on a platform called "Windows". Peer B merely has the **NfpService** initialized, and it is running on a platform called "Android".

Peer A's **NfpService** happens to have a **SourceID** = 0x80, 0x29, 0x84, 0xF4, 0xD6, 0x0E, 0x8D, 0x2B. The base64 encoding for this **SourceID** is "gCmE9NYOjSs".

Peer B's **NfpService** happens to have a **SourceID** = 0xF3, 0x88, 0xC0, 0x6B, 0xE9, 0xCF, 0xD4, 0xDE. The base64 encoding for this **SourceID** is "84jAa+nP1N4".

In this example, the application is uniquely identified on each platform by the following IDs:

- "Android" – "Contoso-Adventure Works-3/6/2012"
- "Windows" – "Contoso%AdventureWorksApp"
- "WinPhone" – "{8342DF32-AD41-8993-927F-CACE4A295751}

Initially, Peer A's **Session Factory** object has the following abstract data model elements:

SessionList: Empty, no **Session** objects (section 3.1.1.4) that represent connections to Peer B.

AppID: "Windows" – "Contoso%AdventureWorksApp"

AlternateIDList: "Android" – "Contoso-Adventure Works-3/6/2012", "WinPhone" – "{8342DF32-AD41-8993-927F-CACE4A295751}

Launch: TRUE.

SessionFactoryID: 0x6c331689, c15ca44b. This number is specified as random. However, for this example, this number was specifically chosen to make it easy to recognize.

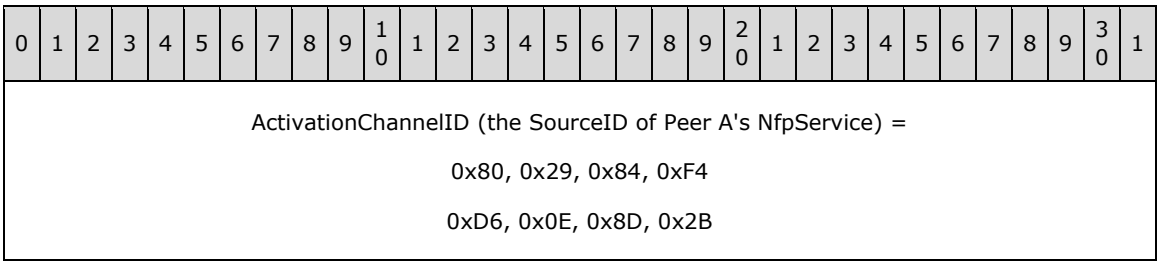
TcpPort: 55555.

RfcommPort: 5.

ReferenceCount: 1 (referenced by the running Adventure Works app).

4.1 Transport Activation and Initial Service Descriptor

When the underlying transport is activated, the protocol begins transmitting. Peer A begins by publishing its pre-initialized **Service Descriptor** message (section 2.2.8) on the well-known channel: "Windows.windows.com/SD". Length = 56 bytes.



...	
OOB Connector ServiceActivationUUID = {E46EDA50-9B5D-41F1-B89E-327B5EA38B16} 0x50, 0xDA, 0x6E, 0xE4 0x5D, 0x9B, 0xF1, 0x41 0xB8, 0x9E, 0x32, 0x7B 0x5E, 0xA3, 0x8B, 0x16	
...	
...	
ExtendedInfo1 = 0x00, 0x00	ServiceVersion = 0x00, 0x01
ExtendedInfo2 = 0x00, 0x00	ExtendedPayloadLength = 0x00, 0x00
Session Factory ServiceActivationUUID: {F1DEBC56-CFBA-4129-983B-7D79499D1A7D} 0x56, 0xBC, 0xDE, 0xF1 0xBA, 0xCF, 0x29, 0x41 0x98, 0x3B, 0x7D, 0x79 0x49, 0x9D, 0x1A, 0x7D	
...	
...	
ExtendedInfo1 = 0x00, 0x00	ServiceVersion = 0x00, 0x01
ExtendedInfo2 = 0x00, 0x00	ExtendedPayloadLength = 0x00, 0x00

4.2 Peer A Service Descriptor Received by Peer B

When a valid **Service Descriptor (SD)** structure (section 2.2.9) is received by Peer B, it will immediately respond with its own **Service Descriptor** message (section 2.2.8) on the same well-known channel: "Windows.windows.com/SD". Length is equal to 56 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ActivationChannelID (the SourceID of Peer B's NfpService) = 0xF3, 0x88, 0xC0, 0x6B 0xE9, 0xCF, 0xD4, 0xDE																															
...																															
Session Factory ServiceActivationUUID: {F1DEBC56-CFBA-4129-983B-7D79499D1A7D} 0x56, 0xBC, 0xDE, 0xF1 0xBA, 0xCF, 0x29, 0x41 0x98, 0x3B, 0x7D, 0x79 0x49, 0x9D, 0x1A, 0x7D																															
...																															
...																															
ExtendedInfo1 = 0x00, 0x00																				ServiceVersion = 0x00, 0x01											
ExtendedInfo2 = 0x00, 0x00																				ExtendedPayloadLength = 0x00, 0x00											
OOB Connector ServiceActivationUUID = {E46EDA50-9B5D-41F1-B89E-327B5EA38B16} 0x50, 0xDA, 0x6E, 0xE4 0x5D, 0x9B, 0xF1, 0x41 0xB8, 0x9E, 0x32, 0x7B 0x5E, 0xA3, 0x8B, 0x16																															
...																															
...																															
ExtendedInfo1 = 0x00, 0x00																				ServiceVersion = 0x00, 0x01											
ExtendedInfo2 = 0x00, 0x00																				ExtendedPayloadLength = 0x00, 0x00											

In this example, Peer B also responds to the **SD** with the **OOB Connector Service Activation** message (section 2.2.5) on Peer A's **Service Activation** base64-encoded **SourceID**: "Windows.gCmE9NYOjSs". Length is equal to 186 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SourceID (of Peer B's NfpService) = 0xF3, 0x88, 0xC0, 0x6B 0xE9, 0xCF, 0xD4, 0xDE																															
...																															
OOB Connector ServiceActivationUUID = {E46EDA50-9B5D-41F1-B89E-327B5EA38B16} 0x50, 0xDA, 0x6E, 0xE4 0x5D, 0x9B, 0xF1, 0x41 0xB8, 0x9E, 0x32, 0x7B 0x5E, 0xA3, 0x8B, 0x16																															
...																															
...																															
ExtendedInfo = 0x00, 0x00																ServiceVersion = 0x00, 0x01															
ReplyChannelID (the OOBConnectorID of the newly created OOB Connector object) = "bcso+pFofkc" 0x6D, 0xCB, 0x28, 0xFA 0x91, 0x68, 0x7E, 0x47																															
...																															
WiFiDirectAddress = fe80::c8:b1:5d9d:779e:81b2 0xFE, 0x80, 0x00, 0x00 0x00, 0x00, 0x00, 0x00 0xC8, 0xB1, 0x5D, 0x9D 0x77, 0x9E, 0x81, 0xB2																															
...																															
...																															
LinkLocalAddress = fe80::3858:bb83:6ca5:11b8																															
...																															
...																															

IPv4LinkLocalAddress = 172.31.233.146 (::ffff:ac1f:e992) 0x00, 0x00, 0x00, 0x00 0x00, 0x00, 0x00, 0x00 0x00, 0x00, 0xFF, 0xFF 0xAC, 0x1F, 0xE9, 0x92
...
...
ProximityAddress = ::
...
...
GlobalAddress = 2001:4898:001a:0003:3858:bb83:6ca5:11b8
...
...
TeredoAddress (16 bytes)
...
...
...
Reserved = 0x00, 0x00, 0x00, 0x00
BluetoothMACAddress = e0:ca:94:49:33:34 0x34, 0x33, 0x49, 0x94 0xCA, 0xE0, 0x00, 0x00
...

WiFiDirectConnectBlobLength = 0x00, 0x28	WiFiDirectConnectBlob = 0x28, 0x00

WiFiDirectConnectBlob (continued) = 0x02, 0x00, 0x10, 0x02 0x01, 0x1F, 0x00, 0x12 0x0C, 0xE3, 0x6E, 0x57 0xE2, 0x01, 0x88, 0x00 0x01, 0x00, 0x50, 0xF2 0x00, 0x00, 0x00, 0x24 0x10, 0x11, 0x00, 0x0A 0x54, 0x52, 0x41, 0x56 0x4D, 0x2D, 0x4E, 0x49 0x4B, 0x45
...

4.3 Peer B Service Descriptor Received by Peer A

When Peer B's valid **Service Descriptor (SD)** structure (section 2.2.9) is received by Peer A, it responds with a **Session Factory Service Activation** message (section 2.2.12) on Peer B's **Service Activation** base64-encoded **SourceID**: "Windows.84jAa+nP1N4". Length is equal to 141 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SourceID (of Peer A's NfpService) = 0x80, 0x29, 0x84, 0xF4 0xD6, 0x0E, 0x8D, 0x2B
...
Session Factory ServiceActivationUUID: {F1DEBC56-CFBA-4129-983B-7D79499D1A7D}
...
...
ExtendedInfo = 0x00, 0x00	ServiceVersion = 0x00, 0x01

ReplyChannelID (the SessionFactoryID of the Adventure Works App's Session Factory) = "bDMWicFcpEs" 0x6C, 0x33, 0x16, 0x89 0xC1, 0x5C, 0xA4, 0x4B		
...		
ClientPreference = 0x00, 0x01, 0x00, 0x00		
AppInfoCount = 0x0003	PlatformQualifierSize = 0x07	PlatformQualifier = "Windows" 0x57, 0x69, 0x6E, 0x64, 0x6F, 0x77, 0x73
...		
...		
AppIDSize = 0x19	AppID = "Contoso%AdventureWorksApp"	
...		
...		
PlatformQualifierSize = 0x07	PlatformQualifier = "Android"	
...		
...		
AppIDSize = 0x20	AppID = "Contoso-Adventure Works-3/6/2012"	
...		
...		
PlatformQualifierSize = 0x08	PlatformQualifier = "WinPhone"	
...		
...		
AppIDSize = 0x26	AppID = "{8342DF32-AD41-8993-927F-CACE4A295751}"	
...		
...		

4.4 Peer A Receives OOB Connector Service Activation Message, Responds with OOB Connector Service ACK

When Peer B's valid **OOB Connector Service Activation** message (section 2.2.5) is received by Peer A, it responds with a **OOB Connector Service ACK** message (section 2.2.4) on the received **ReplyChannelID**, which is the **OOBConnectorID** of Peer B's newly created **OOB Connector** object (section 3.1.1.2): "Windows.bcso+pFofkc". Length is equal to 106 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WiFiDirectAddress = fe80:0000:0000:0000:0dd5:fba4:be61:fedf																															
...																															
...																															
LinkLocalAddress = fe80:0000:0000:0000:a87f:8ed4:32c2:a4dd																															
...																															
...																															
IPv4LinkLocalAddress = 172.31.233.149 (0000:0000:0000:0000:0000:ffff:ac1f:e995)																															
...																															
...																															
ProximityAddress (16 bytes)																															
...																															
...																															
...																															
GlobalAddress (16 bytes)																															
...																															
...																															
...																															
TeredoAddress (16 bytes)																															
...																															

...
...
BluetoothMACAddress = (00:19:0e:08:6f:8f) 0x8F, 0x6F, 0x08, 0x0E 0x19, 0x00, 0x00, 0x00
...
WiFiDirectListenBlobLength = 0x0000

4.5 Peer A Session Factory Service Activation Received by Peer B, Responds with Session Activation

When Peer A's valid **Session Factory Service Activation** message (section 2.2.12) is received by Peer B, it responds by launching the Adventure Works App, creating a **Session Factory** object, creating a **Session** object (section 3.1.1.4), and replying with a **Session Activation** message (section 2.2.11) on the received **ReplyChannelID**, which is the **SessionFactoryID** of Peer A's **Session Factory** object (section 3.1.1.3) for the Adventure Works application: "Windows.bDMWicFcpEs". Length is equal to 96 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SourceID (the SourceID of Peer B's NfpService) = 0xF3, 0x88, 0xC0, 0x6B 0xE9, 0xCF, 0xD4, 0xDE																															
...																															
ActivatedSessionFactoryID (newly created Session Factory object's SessionFactoryID) = 0x40, 0xCA, 0xDB, 0x31 0x50, 0x96, 0xD8, 0x32																															
...																															
ReplyChannelID (newly created Session object's SessionID) = "rhlJshr/7Ew" 0xAE, 0x19, 0x49, 0xB2 0x1A, 0xFF, 0xEC, 0x4C																															
...																															
ECDHPublicKeyMagicNumber = 0x45, 0x43, 0x4B, 0x31																															

ECDHPublicKeyLength (little-endian) = 0x20, 0x00, 0x00, 0x00
ECDHXParam
ECDHYParam

4.6 Peer B Session Activation Received by Peer A, Responds with Session ACK

When Peer B's valid **Session Activation** message (section 2.2.11) is received by Peer A, it responds with a **Session ACK** message (section 2.2.10) on the received **ReplyChannelID,** which is the **SessionID** of Peer B's newly created **Session** object (section 3.1.1.4): "Windows.rhlJshr/7Ew". Length is equal to 76 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ECDHPublicKeyMagicNumber = 0x45, 0x43, 0x4B, 0x31																															
...																															
ECDHPublicKeyLength (little-endian) = 0x20, 0x00, 0x00, 0x00																															
ECDHXParam																															
ECDHYParam																															
TCPPort = 51351 (0xC8, 0x97)																RFCOMMPort = 0x01										Reserved1 = 0x00			

4.7 Peer A Session ACK Received by Peer B, Begins Connection Validation

When Peer A's valid **Session ACK** message (section 2.2.10) is received by Peer B, it responds with an **Accept Header** (section 2.2.1) on the **SessionID** of its **Session** object (section 3.1.1.4): "Windows.rhlJshr/7Ew". The **ConnectionType** field indicates that this is an IPv4 connection. This begins the connection validation handshake.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SessionID = "rhlJshr/7Ew" 0xAE, 0x19, 0x49, 0xB2 0x1A, 0xFF, 0xEC, 0x4C																															
...																															

ConnectionType **=** Link Local (IPv4)
0x00, 0x00, 0x00, 0x00
0x00, 0x00, 0x00, 0x02

4.8 Peer B Accept Header Received by Peer A, Completes Connection Validation

When Peer B's valid **Accept Header** (section 2.2.1) is received by Peer A, it responds by saving the structure in its **NfpService** state element **HandshakeData** and returning the identical **Accept Header** shown in section 4.7. This completes the connection validation handshake.

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 (Updated Section) Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system
- Windows Server 2016 operating system
- Windows Server 2019 operating system
- Windows Server 2022 operating system
- Windows 11 operating system
- Windows Server 2025 operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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.

<1> Section 2.2.6: Windows 8 and Windows Server 2012: Role compatibility checking is not supported.

<2> Section 2.2.11~~<2> Section 2.2.11:~~: Windows 8 and Windows Server 2012: The **Role** field in the **Session Factory Service Activation** message is not supported.

<3> Section 2.2.12: Windows 8 and Windows Server 2012: The host or client **ServiceActivationUUID** value is not supported.

<4> Section 2.2.12: Windows 8 and Windows Server 2012: The **Role** field is not supported.

<5> Section 3.1.2: Windows: The default value is 10 seconds.

<6> Section 3.1.2: Windows: The default value is 10 seconds.

<7> Section 3.1.2: Windows: A timeout value of 10 minutes is used.

<8> Section 3.1.2: Windows: A timeout value of 6 minutes is used.

<9> Section 3.1.5.6: Windows 8 and Windows Server 2012: The **Role** field in the **Session Factory Service Activation** message is not supported.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
6 Appendix A: Product Behavior	Added Windows Server 2025 to the list of applicable products.	Major

8 Index

P

R

S

T

V