

Web Services on Devices

Article • 06/19/2021

Purpose

The Microsoft Web Services on Devices API (WSDAPI) supports the implementation of client-controlled devices and services, and device hosts conforming to the [Devices Profile for Web Services](#) [↗](#) (DPWS). WSDAPI uses [WS-Discovery](#) [↗](#) for device discovery.

WSDAPI may be used for the development of both client and service implementations.

Where applicable

Web Services on Devices allows a client to discover and access a remote device and its associated services across a network. It supports device discovery, description, control, and eventing. Developers can create WSDAPI client proxies and corresponding stubs for device hosts.

Developer audience

The Web Services on Devices documentation is intended for C/C++ programmers and device vendors creating DPWS-compliant products.

Run-time requirements

Client applications that use WSDAPI are supported starting with Windows Vista and Windows Server 2008.

In this section

[↗](#) Expand table

Topic	Description
About Web Services on Devices	Architectural and general information about Web Services on Devices.
Using Web Services on Devices	Information about generating code, configuring applications, and troubleshooting.

Topic	Description
Web Services on Devices Reference	Reference documentation for the Web Services on Devices API.

Related topics

[Dan Driscoll's Blog](#)

About Web Services on Devices

Article • 02/10/2021

Web Service on Devices API (WSDAPI) is an implementation of the [Devices Profile for Web Services](#) (DPWS) for Windows Vista and Windows Server 2008. The DPWS constrains Web Services specifications so clients can easily discover devices. Once a device is discovered, a client can retrieve a description of services hosted on that device and use those services.

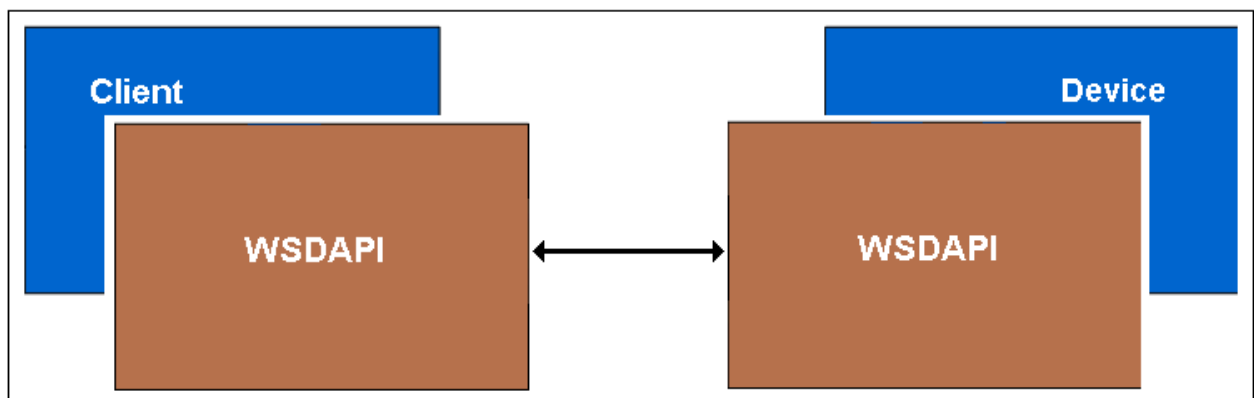
Devices and Services

Devices are components, usually hardware, which are attached to the network. Examples include printers, Web cameras, and video systems.

Devices may include zero or more *services*. For example, a video device may include services that support power on and off, play control, media ejection, and video streaming. Play control may support actions such as play, pause, rewind, and fast forward.

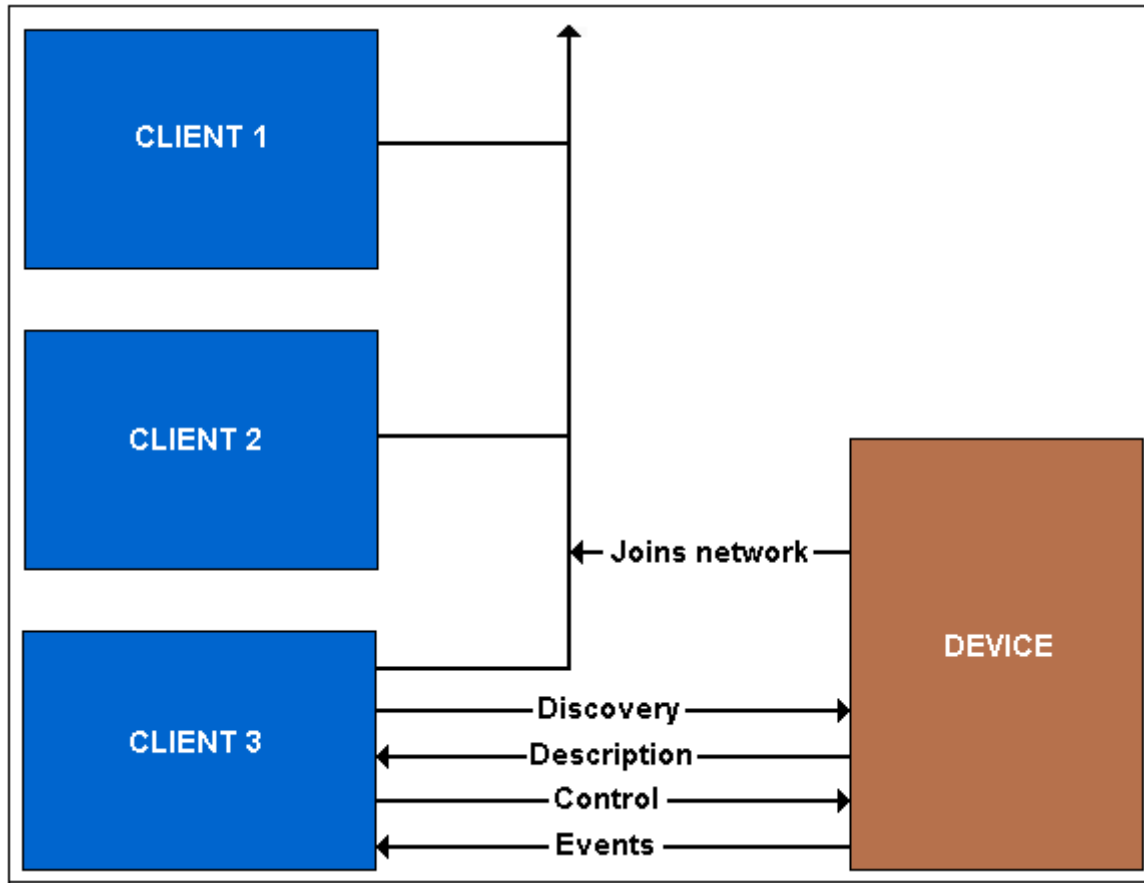
Discovering and Manipulating a Device

WSDAPI extends the local Plug and Play model by allowing a client to discover and access a remote device and its associated services across a network. It supports discovery, one-way and two-way control messaging, and eventing.



DPWS devices announce their presence and expose services (if any) using a unique address and a standardized set of XML messages. DPWS clients can use the discovery process to find the device, enumerate its services, and connect to those services to perform specific actions.

A WSDAPI client first queries the device for complete descriptions of its services, including the service types (such as a printer service type or a scanner service type). The client then controls the device by calling commands defined by a service type (for example, by calling **CreatePrintJob** on a device with a printer service type). Optionally, the client can also monitor state changes in each service by subscribing to events that occur during command execution.



For more information about device messaging patterns, see [Discovery and Metadata Exchange Message Patterns](#).

Logical and Physical Addressing

Logical addressing is used to uniquely identify devices independent of their physical addresses. WS-Discovery provides a mechanism to resolve logical addresses into physical addresses, allowing client-to-device messaging to take place. An example is network attached storage (NAS) that you carry with you. If you have a laptop and a NAS, your laptop should be able to recognize that it is the same device, regardless of the physical address (IP address) that the NAS obtains as you move between subnets. Accomplishing this requires the device have identity that is independent of the IP address it obtains; since traditional mechanisms like DNS are not available in a normal roaming scenario, WS-Addressing and WS-Discovery provide logical addressing and resolution as an ad-hoc alternative.

When a device is manufactured, it is given a globally unique identifier, represented as a UUID URI. This identifier will never change for the device. When the device is powered on, it will always announce its logical address via a WS-Discovery [Hello](#) message, and will accept requests to convert that to a physical address (typically HTTP) via WS-Discovery [Resolve](#) or [Probe](#) messages. Once a valid physical address (IP address) is obtained, all messaging happens over that address, and WS-Discovery is used only if the address changes, the device changes state and the clients need to be notified, or the device goes offline.

Building Applications

WSDAPI provides a generic DPWS SOAP stack for use by client and service applications. The [Web Services on Devices Code Generator](#) (WsdCodeGen.exe) can be used to convert a service description (WSDL) into proxy and stub code that applications can call directly. This generated code automatically transforms function calls and parameters into SOAP messages and XML fields, and then calls into WSDAPI to issue requests to the remote device or client.

Function Discovery can be used when building WSDAPI applications to create and activate function instances returned by PnP. These function instances contain data that can be used to obtain more information through the PnP APIs when more than just simple discovery is required. For more information, see [Function Discovery](#) and [PnP-X](#).

Related topics

[Discovery and Metadata Exchange Message Patterns](#)

[WSDAPI Specification Compliance](#)

[Overview of the WSDAPI Interfaces](#)

Feedback

Was this page helpful?

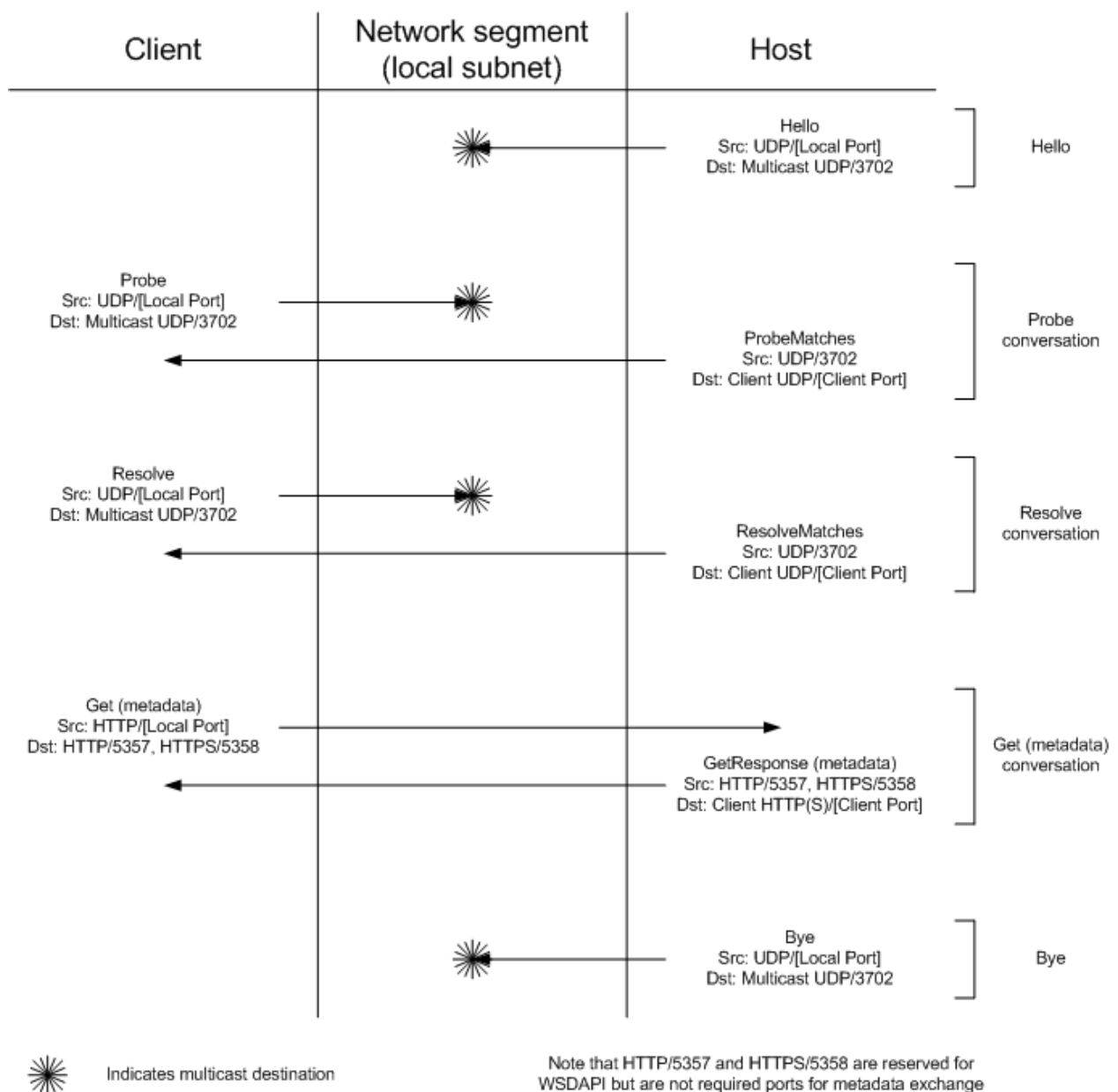
[Get help at Microsoft Q&A](#)

Discovery and Metadata Exchange Message Patterns

Article • 02/10/2021

Device Profile for Web Services (DPWS) hosts and clients communicate over the network using a series of SOAP messages over UDP and HTTP.

The following diagram shows an overview of the expected UDP and HTTP traffic between a DPWS host and client.



[Hello](#), [Bye](#), [Probe](#), [Resolve](#), and [Get](#) messages are all generated without network solicitation; these messages are used to announce device state or to issue a search request. [ProbeMatches](#), [ResolveMatches](#), and [GetResponse](#) messages are generated in response to Probe, Resolve and Get messages.

[Hello](#), [Bye](#), [Resolve](#), and [ResolveMatches](#) messages will always occur over UDP. Similarly, [Get](#) and [GetResponse](#) metadata messages will always occur over HTTP or HTTPS. [Probe](#) and [ProbeMatches](#) messages are normally transmitted over UDP, but take place over an HTTP or HTTPS connection in a directed discovery scenario. For more information about directed discovery message patterns, see [Troubleshooting Applications Using Directed Discovery](#).

The following list shows the typical sequence of messages on the wire. Not all messages are mandatory.

1. [Hello](#)
2. [Probe](#)
3. [ProbeMatches](#)
4. [Resolve](#)
5. [ResolveMatches](#)
6. [Get](#) (metadata exchange request)
7. [GetResponse](#)
8. [Bye](#)

Related topics

[About Web Services on Devices](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Hello Message

Article • 07/13/2023

A Hello message is a WS-Discovery message used to announce the presence of a device or service on the network. Hello messages are also sent in other scenarios. For more information about Hello messages, see section 4.1 of the [WS-Discovery Specification](#).

A Hello message is sent by UDP multicast to port 3702. This message is unsolicited.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the **WSDAPI Basic Interoperability Tool (WSDBIT)** instead.

The following SOAP message shows a sample Hello message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery"
  xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/devprof">
  <soap:Header>
    <wsa:To>
      urn:schemas-xmlsoap-org:ws:2005:04:discovery
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2005/04/discovery/Hello
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:0f5d604c-81ac-4abc-8010-51dbffad55f2
    </wsa:MessageID>
    <wsd:AppSequence InstanceId="2"
      SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772"
      MessageNumber="14">
    </wsd:AppSequence>
  </soap:Header>
  <soap:Body>
    <wsd:Hello>
      <wsa:EndpointReference>
        <wsa:Address>
```

```

urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
</wsa:Address>
</wsa:EndpointReference>
<wsd:Types>wsdp:Device</wsd:Types>
<wsd:MetadataVersion>2</wsd:MetadataVersion>
</wsd:Hello>
</soap:Body>

```

A Hello message has the following focus points.

Focus point	XML	Description
Hello	<pre> <wsa:Action> https://schemas.xmlsoap.org/ws/2005/04/discovery/Hello </wsa:Action> </pre>	The Hello SOAP action identifies the message as a Hello message.
AppSequence	<pre> <wsd:AppSequence InstanceId="2" SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772" MessageNumber="14"> </wsd:AppSequence> </pre>	Contains application sequencing information, which helps to maintain the sequence of messages even if they are received out of order. The AppSequence is validated as described in AppSequence Validation Rules .
Address	<pre> <wsa:Address> urn:uuid:37f86d35-e6ac- </pre>	Contains the endpoint address. This address may be referenced in a Resolve message.

Focus point	XML	Description
	<pre>4241-964f- 1d9ae46fb366 </wsa:Address ></pre>	
Types	<pre><wsd:Types>ws dp:Device</pre>	Contains the WS-Discovery types advertised by the host.

Related topics

[Discovery and Metadata Exchange Messages](#)

[Bye Message](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Probe Message

Article • 08/25/2021

A Probe message is a WS-Discovery message used by a client to search for services on the network by service type. For more information about Probe messages, see section 5.2 of the [WS-Discovery Specification](#).

A Probe message is sent by UDP multicast to port 3702. Unicast Probe messages are not supported.

DPWS clients send Probe messages. The following list shows scenarios in which WSDAPI will send a Probe message.

- Function Discovery clients send Probe messages.
- WSDAPI clients calling [IWSDDiscoveryProvider::SearchByAddress](#) send Probe messages.
- WSDAPI clients calling [IWSDDiscoveryProvider::SearchByType](#) send Probe messages.
- Applications using directed discovery send Probe messages over HTTP or HTTPS.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) instead.

The following SOAP message shows a sample Probe message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery"
  xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/devprof">
  <soap:Header>
    <wsa:To>
      urn:schemas-xmlsoap-org:ws:2005:04:discovery
    </wsa:To>
    <wsa:Action>
```

```

    https://schemas.xmlsoap.org/ws/2005/04/discovery/Probe
  </wsa:Action>
  <wsa:MessageID>
    urn:uuid:29cf10da-5c41-4d55-b184-5ee15e38ce23
  </wsa:MessageID>
</soap:Header>
<soap:Body>
  <wsd:Probe>
    <wsd:Types>wsdp:Device</wsd:Types>
  </wsd:Probe>
</soap:Body>

```

A Probe message has the following focus points.

Focus point	XML	Description
Probe	<pre> <wsa:Action> https://schemas.xmlsoap.org/ws/2005/04/discovery/Probe </wsa:Action> </pre>	The Probe SOAP action identifies the message as a Probe message.
MessageID	<pre> <wsa:MessageID> urn:uuid:29cf10da-5c41-4d55-b184-5ee15e38ce23 </wsa:MessageID> </pre>	Contains the message identifier, which is referenced by the RelatesTo element in a ProbeMatches message.
Types	<pre> <wsd:Types>wsdp:Device </pre>	Contains the WS-Discovery types for which the client is searching. This element should not be empty.

Related topics

[Discovery and Metadata Exchange Messages](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at [Microsoft Q&A](#)

ProbeMatches Message

Article • 07/13/2023

A ProbeMatches message is a WS-Discovery message sent by a service in response to a client's [Probe](#) message. For more information about ProbeMatches messages, see section 5.3 of the [WS-Discovery Specification](#) [↗](#).

A ProbeMatches message is sent by UDP unicast to the port from which the client's [Probe](#) message was sent. ProbeMatches must be sent within 4 seconds of the Probe message; otherwise, Windows Firewall may drop the packet.

If no XAddrs are included in the ProbeMatches message, then the client may send a [Resolve](#) message by UDP multicast to port 3702. The client will only send a Resolve message when an HTTP message (such as a [Get](#) metadata exchange request or a service message) will be sent.

Any DPWS application that sends [Probe](#) messages will receive ProbeMatches messages.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) [↗](#) instead.

The following SOAP message shows a sample ProbeMatches message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery"
  xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/devprof">
  <soap:Header>
    <wsa:To>
      https://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches
    </wsa:Action>
    <wsa:MessageID>
```

```

urn:uuid:967d0036-fe69-40ad-8191-dd1fc8ef64ab
</wsa:MessageID>
<wsa:RelatesTo>
  urn:uuid:29cf10da-5c41-4d55-b184-5ee15e38ce23
</wsa:RelatesTo>
<wsd:AppSequence InstanceId="1"
  SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772"
  MessageNumber="9">
</wsd:AppSequence>
</soap:Header>
<soap:Body>
  <wsd:ProbeMatches>
    <wsd:ProbeMatch>
      <wsa:EndpointReference>
        <wsa:Address>
          urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
        </wsa:Address>
      </wsa:EndpointReference>
      <wsd:Types>wsdp:Device</wsd:Types>
      <wsd:XAddrs>
        https://192.168.0.2:5357/37f86d35-e6ac-4241-964f-
1d9ae46fb366
      </wsd:XAddrs>
      <wsd:MetadataVersion>2</wsd:MetadataVersion>
    </wsd:ProbeMatch>
  </wsd:ProbeMatches>
</soap:Body>
</soap:Envelope>

```

A ProbeMatches message has the following focus points.

Focus point	XML	Description
ProbeMatches	<pre> <wsa:Action> https://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches </wsa:Action> </pre>	The ProbeMatches SOAP action identifies the message as a ProbeMatches message.
RelatesTo	<pre> <wsa:RelatesTo> urn:uuid:29cf </pre>	The identifier of the message to which the service is responding. This header matches the MessageId in the Probe message.

Focus point	XML	Description
	<pre> 10da-5c41- 4d55-b184- 5ee15e38ce23 </wsa:Relates To> </pre>	
AppSequence	<pre> <wsd:AppSeque nce InstanceId="1 " SequenceId="u rn:uuid:369a7 d7b-5f87- 48a4-aa9a- 189edf2a8772" MessageNumber ="9"> </wsd:AppSequ ence> </pre>	Contains application sequencing information, which helps to maintain the sequence of messages even if they are received out of order. The AppSequence is validated as described in AppSequence Validation Rules .
Address	<pre> <wsa:Address> urn:uuid:37f8 6d35-e6ac- 4241-964f- 1d9ae46fb366 </wsa:Address > </pre>	Contains the endpoint address. This address may be referenced in a Resolve message.
XAddr	<pre> <wsd:XAddr> https://192.1 68.0.2:5357/3 7f86d35-e6ac- 4241-964f- 1d9ae46fb366 </wsd:XAddr> </pre>	XAddrs are transport addresses that may be used for communication between client and service. Addrs are validated as described in XAddr Validation Rules .

Related topics

[Discovery and Metadata Exchange Messages](#)

[Probe Message](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Resolve Message

Article • 08/25/2021

A Resolve message is a WS-Discovery message used by a client to search for services on the network by name. A client will only send a Resolve message when an HTTP message (such as a [Get](#) metadata exchange request or a service message) will be sent. For more information about Resolve messages, see section 6.1 of the [WS-Discovery Specification](#) [↗](#).

A Resolve message is sent by UDP multicast to port 3702. Unicast Resolve messages are not supported.

DPWS clients send Resolve messages. The following list shows scenarios in which WSDAPI will send a Resolve message.

- A Function Discovery client sends a Resolve message if no XAddrs are included in a [ProbeMatches](#) message.
- A client calling the [IWSDDiscoveryProvider::SearchById](#) methods will send a Resolve message.
- A client calling [WSDCreateDeviceProxy](#) may send a Resolve message if a logical device address is passed to *pszDeviceId*.
- A client calling [WSDCreateDeviceProxyAdvanced](#) will send a Resolve message if the function is called with the *pDeviceAddress* parameter set to **NULL**.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) [↗](#) instead.

The following SOAP message shows a sample Resolve message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery">
  <soap:Header>
```

```

    <wsa:To>
urn:schemas-xmlsoap-org:ws:2005:04:discovery
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2005/04/discovery/Resolve
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:38d1c3d9-8d73-4424-8861-6b7ee2af24d3
    </wsa:MessageID>
  </soap:Header>
  <soap:Body>
    <wsd:Resolve>
      <wsa:EndpointReference>
        <wsa:Address>
          urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
        </wsa:Address>
      </wsa:EndpointReference>
    </wsd:Resolve>
  </soap:Body>
</soap:Envelope>

```

A Resolve message has the following focus points.

Focus point	XML	Description
Resolve	<pre> <wsa:Action> https://schemas.xmlsoap.org /ws/2005/04/discovery/Resolve </wsa:Action> </pre>	The Resolve SOAP action identifies the message as a Resolve message.
MessageID	<pre> <wsa:MessageID> urn:uuid:38d1c3d9-8d73- 4424-8861-6b7ee2af24d3 </wsa:MessageID> </pre>	Contains the message identifier, which is referenced in a ResolveMatches message.

Focus point	XML	Description
Address	<pre><wsa:Address> urn:uuid:37f86d35-e6ac- 4241-964f-1d9ae46fb366 </wsa:Address></pre>	Contains the address of the endpoint being resolved.

Related topics

[Discovery and Metadata Exchange Messages](#)

[ResolveMatches Message](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

ResolveMatches Message

Article • 07/13/2023

A ResolveMatches message is a WS-Discovery message sent in response to a client's [Resolve](#) message by a matching service. For more information about ResolveMatches messages, see section 6.2 of the [WS-Discovery Specification](#).

A ResolveMatches message is sent by UDP unicast to port 3702 (the port from which the client's [Resolve](#) message was sent). ResolveMatches must be sent within 4 seconds of the Resolve message; otherwise, Windows Firewall may drop the packet.

Any DPWS application that sends [Resolve](#) messages will receive ResolveMatches messages.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) instead.

The following SOAP message shows a sample ResolveMatches message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery"
  xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/devprof">
  <soap:Header>
    <wsa:To>
      https://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2005/04/discovery/ResolveMatches
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:64ddd01c-b0d6-4afd-aba6-6f1f161ce9d4
    </wsa:MessageID>
    <wsa:RelatesTo>
      urn:uuid:38d1c3d9-8d73-4424-8861-6b7ee2af24d3
    </wsa:RelatesTo>
```

```

    <wsd:AppSequence InstanceId="1"
      SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772"
      MessageNumber="6">
    </wsd:AppSequence>
  </soap:Header>
  <soap:Body>
    <wsd:ResolveMatches>
      <wsd:ResolveMatch>
        <wsa:EndpointReference>
          <wsa:Address>
            urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
          </wsa:Address>
        </wsa:EndpointReference>
        <wsd:Types>wsdp:Device</wsd:Types>
        <wsd:XAddrs>
          https://192.168.0.2:5357/37f86d35-e6ac-4241-964f-
1d9ae46fb366
        </wsd:XAddrs>
        <wsd:MetadataVersion>2</wsd:MetadataVersion>
      </wsd:ResolveMatch>
    </wsd:ResolveMatches>
  </soap:Body>
</soap:Envelope>

```

A ResolveMatches message has the following focus points.

Focus point	XML	Description
ResolveMatches	<pre> <wsa:Action> https://schemas.xmlsoap.org/ws/2005/04/discovery/ResolveMatches </wsa:Action> </pre>	The ResolveMatches SOAP action identifies the message as a ResolveMatches message.
RelatesTo	<pre> <wsa:RelatesTo> urn:uuid:38d1c3d9-8d73-4424-8861-6b7ee2af24d3 </pre>	The identifier of the message to which the service is responding. This header matches the MessageId in the Resolve message.

Focus point	XML	Description
	<pre></wsa:Relates To></pre>	
AppSequence	<pre><wsd:AppSeque nce InstanceId="1 " SequenceId="u rn:uuid:369a7 d7b-5f87- 48a4-aa9a- 189edf2a8772" MessageNumber ="6"> </wsd:AppSequ ence></pre>	Contains application sequencing information, which helps to maintain the sequence of messages even if they are received out of order. The AppSequence is validated as described in AppSequence Validation Rules .
Address	<pre><wsa:Address> urn:uuid:37f8 6d35-e6ac- 4241-964f- 1d9ae46fb366 </wsa:Address ></pre>	Contains the address of the endpoint being resolved.
XAddr	<pre><wsd:XAddr> https://192.1 68.0.2:5357/3 7f86d35-e6ac- 4241-964f- 1d9ae46fb366 </wsd:XAddr></pre>	XAddr is transport addresses that may be used for communication between client and service. Addr is validated as described in XAddr Validation Rules .

Related topics

[Discovery and Metadata Exchange Messages](#)

[Resolve Message](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Get (Metadata Exchange) HTTP Request and Message

Article • 07/13/2023

A Get message is a WS-Transfer message used to request metadata. For more information about Get messages, see section 3.1 of the [WS-Transfer Specification](#). Because metadata exchange is done over HTTP, a Get message is the payload of an HTTP request.

DPWS clients send Get messages. Function Discovery clients, WSDAPI clients calling [WSDCreateDeviceProxy](#), and WSDAPI clients calling [WSDCreateDeviceProxyAdvanced](#) send this message.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) instead.

The following example shows a sample Get HTTP request.

syntax

```
POST /37f86d35-e6ac-4241-964f-1d9ae46fb366
HTTP/1.1
Content-Type: application/soap+xml
User-Agent: WSDAPI
Host: 192.168.0.2:5357
Content-Length: 658
Connection: Keep-Alive
Cache-Control: no-cache
Pragma: no-cache
```

A Get HTTP request has the following focus points.

Focus point	Header Line	Description
URL Path		The URL path where the Get HTTP request was posted.

Focus point	Header Line	Description
	POST /37f86d35-e6ac-4241-964f-1d9ae46fb366	
Host and Port	Host: 192.168.0.2:5357	The host and port where the Get HTTP request was directed.

The following SOAP message shows a sample Get message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soap:Header>
    <wsa:To>
      urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:027bec45-c37c-466c-936c-68f648abe2bb
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>
        https://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:From>
      <wsa:Address>
        urn:uuid:49e131df-351a-4ece-9a6f-6a862d31cffa
      </wsa:Address>
    </wsa:From>
  </soap:Header>
  <soap:Body>
  </soap:Body>
```

A Get message has the following focus points.

Focus point	XML	Description
To	<pre> <wsa:To> urn:uuid:37f86d35- e6ac-4241-964f- 1d9ae46fb366 </wsa:To> </pre>	The identifier of the device being asked for metadata.
Get	<pre> <wsa:Action> https://schemas.xmlsoap.o rg/ws/2004/09/transfer/Ge t </pre>	The Get SOAP action identifies the message as a Get message.
MessageID	<pre> <wsa:MessageID> urn:uuid:027bec45- c37c-466c-936c- 68f648abe2bb </pre>	Contains the message identifier, which is referenced in a GetResponse message.

Related topics

[Discovery and Metadata Exchange Messages](#)

[GetResponse Message](#)

Feedback

Was this page helpful?



Yes



No

GetResponse (Metadata Exchange) Message

Article • 08/25/2021

A GetResponse message is a WS-Transfer message used to respond to a request for metadata. For more information about GetResponse messages, see section 3.1 of the [WS-Transfer Specification](#).

Any DPWS application that sends [Get](#) messages will receive GetResponse messages.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) instead.

The following SOAP message shows a sample GetResponse message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsx="https://schemas.xmlsoap.org/ws/2004/09/mex"
  xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/devprof"
  xmlns:sim="https://schemas.example.org/SimpleService"
  xmlns:att="https://schemas.example.org/AttachmentService"
  xmlns:eve="https://schemas.example.org/EventingService">
  <soap:Header>
    <wsa:To>
      https://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:119dcdf5-fc0d-4d40-bf1f-de52dc292744
    </wsa:MessageID>
    <wsa:RelatesTo>
      urn:uuid:027bec45-c37c-466c-936c-68f648abe2bb
    </wsa:RelatesTo>
  </soap:Header>
```

```

<soap:Body>
  <wsx:Metadata>
    <wsx:MetadataSection
Dialect="https://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice">
      <wsdp:ThisDevice>
        <wsdp:FriendlyName>
          WSDAPI Basic Interop Server
        </wsdp:FriendlyName>
        <wsdp:FirmwareVersion>alpha</wsdp:FirmwareVersion>
        <wsdp:SerialNumber>1</wsdp:SerialNumber>
      </wsdp:ThisDevice>
    </wsx:MetadataSection>
    <wsx:MetadataSection
Dialect="https://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel">
      <wsdp:ThisModel>
        <wsdp:Manufacturer>
          Microsoft
        </wsdp:Manufacturer>
        <wsdp:ManufacturerUrl>
          https://www.microsoft.com/
        </wsdp:ManufacturerUrl>
        <wsdp:ModelName>
          WSDAPI Interop device
        </wsdp:ModelName>
        <wsdp:ModelNumber>0.1</wsdp:ModelNumber>
        <wsdp:ModelUrl>
          https://www.microsoft.com/
        </wsdp:ModelUrl>
        <wsdp:PresentationUrl>
          https://www.microsoft.com/
        </wsdp:PresentationUrl>
      </wsdp:ThisModel>
    </wsx:MetadataSection>
    <wsx:MetadataSection
Dialect="https://schemas.xmlsoap.org/ws/2006/02/devprof/Relationship">
      <wsdp:Relationship
Type="https://schemas.xmlsoap.org/ws/2006/02/devprof/host">
        <wsdp:Host>
          <wsa:EndpointReference>
            <wsa:Address>
              urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
            </wsa:Address>
          </wsa:EndpointReference>
          <wsdp:Types>sim:SimpleDeviceType</wsdp:Types>
          <wsdp:ServiceId>
            https://testdevice.interop/SimpleDevice
          </wsdp:ServiceId>
        </wsdp:Host>
        <wsdp:Hosted>
          <wsa:EndpointReference>
            <wsa:Address>
              https://192.168.0.2:5357/37f86d35-e6ac-4241-
964f-1d9ae46fb366
            </wsa:Address>
          </wsa:EndpointReference>

```

```

        <wsdp:Types>sim:SimpleService</wsdp:Types>
        <wsdp:ServiceId>
            https://testdevice.interop/SimpleService1
        </wsdp:ServiceId>
    </wsdp:Hosted>
</wsdp:Relationship>
</wsx:MetadataSection>
</wsx:Metadata>
</soap:Body>
</soap:Envelope>

```

A GetResponse message has the following focus points.

Focus point	XML	Description
GetResponse	<pre> <wsa:Action> https://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse </wsa:Action> </pre>	The GetResponse SOAP action identifies the message as a GetResponse message.
RelatesTo	<pre> <wsa:RelatesTo> urn:uuid:027bec45- c37c-466c-936c- 68f648abe2bb </wsa:RelatesTo> </pre>	The identifier of the message to which the device is responding. This header matches the MessageID in the Get message.
Address	<pre> <wsa:Address> https://192.168.0.2:5 357/37f86d35-e6ac- 4241-964f- 1d9ae46fb366 </wsa:Address> </pre>	Contains the endpoint address of the services hosted on this device.

Related topics

[Discovery and Metadata Exchange Messages](#)

[Get Message](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Bye Message

Article • 07/13/2023

A Bye message is a WS-Discovery message used to announce the departure of a device or service from the network. For more information about Bye messages, see section 4.2 of the [WS-Discovery Specification](#).

Bye messages are unsolicited. The messages are optional.

ⓘ Note

This topic shows a sample DPWS message generated by WSDAPI clients and hosts. WSDAPI will parse and accept other DPWS-compliant messages that do not conform to this sample. Do not use this sample to verify DPWS interoperability; use the **WSDAPI Basic Interoperability Tool (WSDBIT)** instead.

The following SOAP message shows a sample Bye message.

syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
  xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery">
  <soap:Header>
    <wsa:To>
      urn:schemas-xmlsoap-org:ws:2005:04:discovery
    </wsa:To>
    <wsa:Action>
      https://schemas.xmlsoap.org/ws/2005/04/discovery/Bye
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:193ccfa0-347d-41a1-9285-f500b6b96a15
    </wsa:MessageID>
    <wsd:AppSequence InstanceId="2">
      SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772"
      MessageNumber="21">
    </wsd:AppSequence>
  </soap:Header>
  <soap:Body>
    <wsd:Bye>
      <wsa:EndpointReference>
        <wsa:Address>
          urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
```

```
</wsa:Address>
</wsa:EndpointReference>
</wsd:Bye>
</soap:Body>
```

A Bye message has the following focus points.

Focus point	XML	Description
Bye	<pre><wsa:Action> https://schemas.xmlsoap.org/ws/2005/04/discovery/Bye </wsa:Action></pre>	The Bye SOAP action identifies the message as a Bye message.
AppSequence	<pre><wsd:AppSequence InstanceId="2" SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772" MessageNumber="21"> </wsd:AppSequence></pre>	Contains application sequencing information, which helps to maintain the sequence of messages even if they are received out of order. The AppSequence is validated as described in AppSequence Validation Rules .
Address	<pre><wsa:Address> urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366</pre>	Contains the address of the endpoint going offline.

Focus point	XML	Description
	<pre></wsa:Address> ></pre>	

Related topics

[Discovery and Metadata Exchange Messages](#)

[Hello Message](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSDAPI Specification Compliance

Article • 01/07/2021

Web Services on Devices API (WSDAPI) provides support for the [Devices Profile for Web Services](#) [↗] (DPWS) on Windows Vista, which enables Web Services (WS) communication between Windows-based PCs and network connected devices. DPWS assembles basic WS specifications, such as [WS-Eventing](#) [↗], [WS-Discovery](#) [↗], and [WS-MetadataExchange](#) [↗], and enhances or constrains them to provide a baseline set of capabilities for resource-constrained devices. This baseline specification contains required functionality, such as the ability to describe properties of the device through metadata, and optional functionality, such as the ability to reject specific messages for security reasons.

The WSDAPI implementation in Windows Vista is the first release of explicit support for the DPWS, and is an unmanaged implementation of DPWS that is separate and distinct from other Web Services implementations (such as the [Windows Communication Foundation](#) or [WS-Management](#) [↗]).

The following topics are of interest to device manufacturers and other device implementers that create devices that interoperate with Windows-based WSDAPI clients and hosts without using WSDAPI. These topics describe the implementation of WSDAPI relative to the underlying specifications. They cover the implementation of required functionality, optional functionality, and additional functionality not defined in the specification.

- [DPWS Specification Compliance](#)
- [WS-Discovery Specification Compliance](#)
- [WS-MetadataExchange and WS-Transfer Specification Compliance](#)
- [Additional WS-Discovery Functionality](#)

Related topics

[WSD Device Development](#)

[WSD Device Implementation Recommendations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DPWS Specification Compliance

Article • 01/07/2021

This topic describes how WSDAPI implements the elective functionality in the [Devices Profile for Web Services](#) (DPWS) specification. It also describes which DPWS functionality was omitted from the WSDAPI implementation.

The DPWS specification provides a consistent way to message with devices. It also adds specific restrictions and recommendations that simplify the process of supporting web services on embedded hardware.

The DPWS specification describes elective functionality by using the terms MAY or SHOULD in a given implementation recommendation or restriction. Omitted functionality may be functionality described as REQUIRED in the DPWS specification that was not implemented by WSDAPI, or it may be functionality that WSDAPI implemented in a method other in the one specified in the DPWS specification.

This topic follows the layout of the DPWS section by section. Each section describes how specific restrictions, requirements, and elective functionality are handled by the WSDAPI implementation. This topic is best read in tandem with the DPWS specification.

DPWS 3.0 Messaging

DPWS 3.1 URI formats

Restrictions R0025 and R0027 constrain URIs to MAX_URI_SIZE octets. WSDAPI enforces both of these restrictions as specified.

DPWS 3.2 UDP messaging

Recommendation R0029 suggests that UDP packets larger than the maximum transfer unit (MTU) for UDP should not be sent. WSDAPI does not implement this recommendation, and will allow implementations to send and receive discovery messages that are larger than the MTU.

DPWS 3.3 HTTP messaging

R0001 requires that services support chunked transfer. WSDAPI accepts chunked data in request messages, and will send chunked data in request messages.

R0012 and R0013 describe required portions of the SOAP HTTP binding. For R0012, WSDAPI implements the SOAP HTTP binding, but will not begin reading the HTTP response until WSDAPI has finished sending the HTTP request. WSDAPI does implement the required message exchange pattern in R0013, does implement the optional responding SOAP node in R0014, and does not implement the optional web method feature in R0015. WSDAPI also supports the requirements in R0030 and R0017.

DPWS 3.4 SOAP Envelope

WSDAPI supports R0034, and enforces R0003 and R0026 by default. More specifically, in accordance with R0003 and R0026, if WSDAPI receives a SOAP envelope that is larger than MAX_ENVELOPE_SIZE over HTTP it is rejected and the connection is closed.

DPWS 3.5 WS-Addressing

R0004 reflects the recommended use of the device API in WSDAPI, and is supported by the client API in WSDAPI. Since this is a recommendation, WSDAPI will allow clients and devices to use URIs other than `urn:uuid` URIs for their device endpoints to ensure maximum compatibility. Since the device API in WSDAPI does not persist state between initializations, it is up to application developers using the device API in WSDAPI to ensure R0005 and R0006 are properly supported. The client API in WSDAPI will assume that device identities are unique and persisted, and functionality building on the client API in WSDAPI (such as PnP-X) will require this in order to properly recognize the device across device reboots.

R0007 recommends against the use of reference properties in endpoint references. WSDAPI will still recognize and accept endpoints with reference properties, and developers may choose to use them, but by default WSDAPI will not populate them in endpoints it creates. Similarly, with R0042, when WSDAPI creates service endpoints it will use a HTTP or HTTPS transport address, but will not require devices use HTTP or HTTPS transport addresses in their service endpoints. The behavior of the client when attempting to communicate with a service that does not use HTTP or HTTPS is undefined.

On faults, R0031 constrains the reply endpoint and describes the fault to send if the fault is not anonymous. WSDAPI forces the reply endpoint to use the correct value when sending messages, and will fault correctly if WSDAPI receives a request message with the incorrect reply endpoint. R0041 gives implementations the option to drop a fault if the reply endpoint is invalid. Rather than drop the fault, WSDAPI will send the fault back on the request channel, addressed to the anonymous endpoint, as a "best effort" to communicate with the client.

Lastly, there are two restrictions on SOAP headers, R0019 and R0040, both of which WSDAPI complies with and enforces on received messages.

DPWS 3.6 Attachments

WSDAPI supports attachments and complies with R0022. WSDAPI also complies with R0037. When sending attachments, WSDAPI will always set the Content Transfer Encoding to "binary" for all MIME parts. However, WSDAPI does not enforce R0036. The behavior of WSDAPI when receiving a MIME part with a Content Transfer Encoding not set to "binary" is undefined.

DPWS also defines MIME part ordering clauses. For R0038, WSDAPI will enforce part ordering and will reject a MIME message if the SOAP envelope is not the first MIME part. For R0039, WSDAPI will always send the SOAP envelope as the first MIME part.

DPWS 4.0 Discovery

R1013 and R1001 differentiate device discovery and service discovery. WSDAPI complies with R1013. The hosting implementation complies with R1001, but WSDAPI does not enforce this recommendation on the client.

DPWS also provides guidance on types and scope matching rules. WSDAPI supports all of the scope matching rules defined in [WS-Discovery](#) except LDAP. WSDAPI also provides an extensible model for defining custom scope matching rules, thus complying with R1019. The hosting API will also always provide the `wsdp:Device` type in discovery per R1020, but the client API does not require it be present. Other applications built on WSDAPI, such as PnP-X, do have a hard requirement for the `wsdp:Device` type to be present in discovery.

To facilitate discovery and binding, WSDAPI supports R1009 and R1016. Per R1018, WSDAPI will ignore multicast UDP not sent to the anonymous address. R1015, R1021, and R1022 define a HTTP binding for the Probe message, which WSDAPI supports as described.

DPWS 5.0 Description

WSDAPI enforces R2044 on the client. On the hosting side, WSDAPI will only ever provide the `wsx:Metadata` element in the SOAP envelope body. R2045 allows devices to support a subset of the [WS-Transfer](#) functionality. The hosting API will always generate the `wsa:ActionNotSupported` fault.

DPWS 5.1 Characteristics

DPWS describes basic characteristics for the device. In addition to the restrictions described in this topic, length limits are defined for specific strings and URIs. WSDAPI does enforce the length limits in this DPWS section 5.1, either before sending the message or after parsing its contents.

DPWS also describes the required metadata sections and cycling of the metadata version. The client implementation enforces the presence of ThisModel and ThisDevice metadata. The hosting implementation also properly manages the metadata version and always provides these sections, complying with R2038, R2012, R2001, R2039, R2014, and R2002.

DPWS 5.2 Hosting

This section describes the hierarchy of services and relationship metadata. WSDAPI does not enforce the uniqueness of the ServiceId as described in this section on either the client or the device side.

WSDAPI does comply with R2040, and it is possible for the hosting implementation to send a metadata response with no relationship section if there are no hosted services. The client implementation correctly accepts the metadata response.

R2029 allows for multiple relationship sections in a metadata response, which WSDAPI will correctly accept. R2030 and R2042 describe management of the metadata version, which is implemented correctly in the hosting API.

DPWS 5.3 WSDL

If a service provides Web Services Description Language (WSDL) data, client implementations can get the service definition and manipulate the service on the fly. This is used by late bound clients. The WSDAPI client implementation will accept WSDL provided from a service, but the client does not validate it and the client does not provide a late bound programming model. The hosting implementation can be used to provide WSDL, but the host is not required to do so as the service level metadata is not managed by the host itself.

DPWS 5.4 WS-Policy

DPWS describes Policy assertions to be used for devices. Since WSDAPI does not provide and does not interpret WSDL, it cannot recognize and enforce policy embedded

in WSDL data.

DPWS 6.0 Eventing

DPWS 6.1 Subscription

DPWS requires support for Push delivery. WSDAPI implements Push delivery on the service side, thus complying with R3009 and R3010, and will only accept the Push delivery mode on the client side. R3017 and R3018 require specific faults from the service if it does not recognize the `NotifyTo` or `EndTo` addresses. WSDAPI does not validate these addresses upfront and will not generate these faults. However, the client implementation will recognize these faults correctly. Similarly, R3019 is optional and WSDAPI does not implement this recommendation, but the client implementation will correctly recognize the `SubscriptionEnd` message and will notify the application of a delivery failure.

DPWS 6.1.1 Filtering

WSDAPI complies with R3008 and implements the `Action` filter. In compliance with R3011 and R3012, WSDAPI will not generate the faults in the stated conditions. WSDAPI also implements the fault described R3020 if it does not recognize the actions on which it is asked to filter.

DPWS 6.2 Subscription Duration and Renewal

WSDAPI complies with R3005, R3006, and R3016. WSDAPI will always use `xs:duration` but will accept `xs:dateTime` if provided, and thus will not issue the optional fault in R3013. WSDAPI supports `GetStatus` and will not issue the `wsa:ActionNotSupported` fault per R3015. WSDAPI accept the `wsa:ActionNotSupported` fault if a service responds to a `GetStatus` request with it.

DPWS 7.0 Security

DPWS describes a recommended security model for devices. WSDAPI does not implement these recommendations as described, and does not enforce the restrictions in this section as described.

DPWS Appendix I

DPWS amends global constants from other specifications to suit devices. WSDAPI uses the constants from this section, and overrides the default constants in the [WS-Discovery](#) implementation with these constants. Applications using WSDAPI for WS-Discovery will be bound to the constants defined in DPWS, not the constants defined in [WS-Discovery](#).

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

WS-Discovery Specification Compliance

Article • 01/07/2021

[WS-Discovery](#) describes how to perform the following tasks:

- Announce the availability of services on the local subnet
- Search for services on the subnet
- Locate a previously referenced service

To accomplish this, WS-Discovery defines two one-way messages, [Hello](#) and [Bye](#), and two bidirectional search messages, [Probe](#) and [Resolve](#).

WS-Discovery also provides addresses and a reserved port for IPv4 and IPv6 link local discovery. The specification also allows for alternate bindings to be defined elsewhere, such as the Probe over HTTP binding defined in the [Devices Profile for Web Services](#) (DPWS).

The WS-Discovery specification describes elective functionality by using the terms MAY or SHOULD in a given implementation recommendation or restriction. Omitted functionality may be functionality described as REQUIRED in the WS-Discovery specification that was not implemented by WSDAPI, or it may be functionality that WSDAPI implemented in a method other than the one specified in the WS-Discovery specification.

This topic describes how WS-Discovery restrictions, requirements, and elective functionality are handled by the WSDAPI implementation. This topic is best read in tandem with the WS-Discovery specification.

WS-Discovery and SOAP-over-UDP Support

In SOAP-over-UDP, Section 3.2 specifies that the UDP message must fit in a 64K datagram. WSDAPI will accept 64K UDP messages, but the DPWS constraint of MAX_ENVELOPE_SIZE (32K) will constrain the message size. As required by [WS-Discovery](#), WSDAPI supports the message patterns described in Section 4.

WSDAPI may be configured to support the security model in Sections 7 and 8. When so configured, WSDAPI will sign outbound WS-Discovery messages and will validate signatures on inbound messages.

WSDAPI implements the retransmission algorithm defined in Appendix I as amended by DPWS Appendix I.

In WS-Discovery, WSDAPI uses the addresses specified in section 2.4. WSDAPI extends APP_MAX_DELAY from section 2.4, but not to the extent as defined in DPWS Appendix I. For more information about APP_MAX_DELAY, see [Additional WS-Discovery Functionality](#).

WS-Discovery describes the `uuid:` URI format recommendation in section 2.6, but WSDAPI overrides this recommendation. Instead, WSDAPI uses the `urn:uuid:` URI format described in DPWS.

Section 3 of WS-Discovery describes how a client interacts with a discovery proxy. WSDAPI does not recognize this interaction and ignores announcements from discovery proxies. In Windows 7, WSDAPI implements a private extension to the WS-Discovery protocol, WS-Discovery Remote Extensions, to allow discovery clients to search for services spread across many different networks by sending requests to centralized proxies. For more information, see [Additional WS-Discovery Functionality](#).

Section 4.1, paragraph 3 of WS-Discovery requires that a timer must elapse before a [Hello](#) message is sent. The hosting API does not wait before sending a Hello message. If a scenario requires a delay before a Hello message is sent, then the application developer must implement a wait.

WSDAPI implements all of the messages described in WS-Discovery Sections 4, 5, and 6. WSDAPI also enforces the MATCH_TIMEOUT described in Section 7 as amended by DPWS Appendix I. WSDAPI only protects against "Replay" from the secure considerations in Section 9.

WSDAPI does implement application sequencing as described in WS-Discovery Appendix I.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WS-MetadataExchange and WS-Transfer Specification Compliance

Article • 01/07/2021

Before August 2006, [WS-MetadataExchange](#) defined its own [Get](#) metadata exchange method, which was used by [Devices Profile for Web Services](#) (DPWS). The WS-MetadataExchange specification version 1.1 replaced this method with the Get method defined in the WS-Transfer specification.

In the current model, WS-Transfer provides the [Get](#) method and makes no reference to the type of the body. WS-MetadataExchange describes the format of the body and a mechanism for packaging multiple pieces of metadata in a single response, and DPWS describes specific pieces of metadata that a service should include in a metadata response.

WSDAPI does not fully support the WS-Transfer specification. Because only the [Get](#) method is required for devices, no other portions of WS-Transfer have been implemented. Also, WSDAPI does not implement the optional GetMetadata method described in WS-MetadataExchange.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Additional WS-Discovery Functionality

Article • 01/07/2021

In some cases, the [Devices Profile for Web Services](#) (DPWS) and related specifications do not explicitly define implementation functionality. For example, the [WS-Discovery](#) specification does not define client and host behavior in multi-homed environments. When WSDAPI was implemented, some discovery functionality was added beyond the functionality defined in the specification.

WSDAPI also implements selected portions of WS-Discovery v1.1 CD1 for communicating with a discovery proxy over HTTP.

The purpose of this topic is to describe the discovery functionality implemented by WSDAPI but not otherwise described in the DPWS or WS-Discovery specifications.

IPv6 addresses and the soap.udp URI format

SOAP-over-UDP and WS-Discovery do not explicitly describe how a literal IPv6 address is represented in the soap.udp URI format. [RFC 2396](#), entitled "Uniform Resource Identifiers (URI): Generic Syntax", indicates that literal IPv6 addresses are not supported by the soap.udp URI format.

For simplicity, WSDAPI recognizes IPv6 addresses enclosed in square brackets in the soap.udp scheme. For example, the address

`soap.udp://[2001:abcd:0001:0002:0003:0004:0005:0032]:3702` is recognized by WSDAPI.

This is similar to how IPv6 addresses are handled in HTTP.

Hello and XAddrs

WSDAPI's DPWS hosting object will never send a WS-Discovery [Hello](#) message with [XAddrs](#) in the message body. A client will always send a [Resolve](#) message after receiving a Hello message if the client needs to get the XAddrs.

There are two benefits of this approach. First, a device built on WSDAPI will never expose XAddrs that disclose the IP addresses of private networks. Secondly, a device built on WSDAPI only exposes XAddrs that are accessible to the client, which means that IPv6 addresses are never sent to an IPv4 client.

When a [Probe](#) or [Resolve](#) message is received, only a single XAddr is sent in response. The sent XAddr corresponds to the local address on which the request was received. If

the request was received across subnets over IPv6, WSDAPI will provide a global IPv6 address in the response.

Preferred addresses

A device can provide multiple XAddrs in a [Hello](#), [ProbeMatch](#), or [ResolveMatch](#) message. A service can also be available at multiple endpoints with different transport addresses. In these cases, WSDAPI will try to communicate with the device on the first usable address it finds. An address is usable if it is from an available protocol, such as IPv4 on a machine where IPv4 is installed or IPv6 on a machine where IPv6 is installed. Additionally, if the address came from a device or service that is not on the local subnet, it is usable only if it is IPv4, IPv6 site local, or IPv6 link local.

WSDL in metadata exchange

Devices and services built on WSDAPI do not provide their WSDL in metadata exchange unless extended by the application to provide this information. By default, WSDL provision is not part of the programming model.

APP_MAX_DELAY

DPWS defines APP_MAX_DELAY, the random interval to delay between receiving a [Probe](#) and sending a [ProbeMatch](#), as 5,000 milliseconds. Windows Firewall requires that the multicast request/unicast response model for UDP will only work within the 4 second firewall window. As a result, WSDAPI will transmit responses in 2,500 ms or less, instead of the 5,000 ms window described by APP_MAX_DELAY.

IANA port reservations

WSDAPI uses TCP port 5357 for HTTP traffic and TCP port 5358 for HTTPS traffic by default. These ports are reserved for lower privilege processes through a URL reservation in HTTP.sys, and are also reserved with IANA.

UDP port sharing

WSDAPI uses port sharing. Unicast messages sent to port 3702 may not be properly handled by all WSDAPI-based applications. If an application binds exclusively to port 3702, it may prevent WSDAPI-based applications from using that port correctly.

WS-Discovery v1.1 CD1 Proxy

WSDAPI will search for and communicate with a discovery proxy that implements the WS-Discovery v1.1 CD1 managed mode protocol. WS-Discovery v1.1 CD1 is the first revision of WS-Discovery to include an explicit description of an HTTP protocol for communication between a proxy and a client or device.

To limit the number of concurrent versions used in multicast requests, WSDAPI sends a WS-Discovery Probe request in the 2005/04 namespace, but searches for the WS-Discovery v1.1 CD1 DiscoveryProxy type. If a proxy responds, WSDAPI will send an HTTP Probe or Resolve request to the specified proxy endpoint as defined in WS-Discovery v1.1 CD1.

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

XAddr Validation Rules

Article • 01/07/2021

Transport addresses (XAddrs) included in [ProbeMatches](#) and [ResolveMatches](#) messages are subject to basic validation before WSDAPI sends an HTTP message, such as a metadata request.

This is in order to ensure that the XAddrs are on the same subnet as the client.

The following XML shows a sample XAddrs element. The wsd prefix refers to the namespace `https://schemas.xmlsoap.org/ws/2005/04/discovery`.

syntax

```
<wsd:XAddrs>
  https://192.168.0.2:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366
</wsd:XAddrs>
```

All of the following conditions must be met before the HTTP message will go out over the wire.

- XAddrs must be HTTP or HTTPS addresses. XAddrs of other schemes are ignored.
- If any HTTPS XAddrs are present, all XAddrs must be HTTPS. XAddr sections which include both HTTP and HTTPS addresses are completely ignored. Additionally, the device's endpoint address must match the HTTPS XAddrs exactly.
- XAddrs must be IP addresses or hostnames resolvable through DNS. Usually, IP addresses are used.
- At least one IP address included in the XAddrs (or IP address resolved from a hostname included in the XAddrs) must be on the same subnet as the adapter over which the [ProbeMatches](#) or [ResolveMatches](#) message was received.
- The address and port specified in the first XAddr must be accessible. WSDAPI attempts to connect to this address when establishing an HTTP connection.

Related topics

[ProbeMatches](#)

[ResolveMatches](#)

[Discovery and Metadata Exchange Message Patterns](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

AppSequence Validation Rules

Article • 01/07/2021

AppSequence information contained in WS-Discovery announcement and response messages ([Hello](#), [ProbeMatches](#), and [ResolveMatches](#)). This information is processed and validated by WSDAPI before these messages are passed on to components above the stack (such as Network Explorer or an application calling into WSDAPI).

The following XML shows a sample AppSequence element. The wsdl prefix refers to the namespace `https://schemas.xmlsoap.org/ws/2005/04/discovery`.

syntax

```
<wsdl:AppSequence InstanceId="2"
  SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772"
  MessageNumber="21">
</wsdl:AppSequence>
```

WSDAPI ignores stale messages. For each device (uniquely identified by the Endpoint Address in the SOAP Body), WSDAPI ignores any messages with an AppSequence MessageNumber lower than the last message seen.

WSDAPI ignores stale XAddr announcements. If the AppSequence InstanceId is lower than the last InstanceId seen, WSDAPI ignores the XAddrs advertised in the SOAP body. Also, if the InstanceId is the same as previous but the MetadataVersion is lower than the last MetadataVersion, WSDAPI ignores the XAddrs.

WSDAPI ignores duplicate WS-Discovery messages. If two identical WS-Discovery messages are sent to WSDAPI, only the first received will be processed. This is typically only relevant for applications that call directly into the [IWSDDiscoveryPublisher](#) or [IWSDDiscoveryProvider](#) interfaces.

Related topics

[Discovery and Metadata Exchange Message Patterns](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Overview of the WSDAPI Interfaces

Article • 01/07/2021

Web Services on Devices API (WSDAPI) is used to develop client applications that find and access devices, and to develop device hosts and associated services that run on Windows Vista and Windows Server 2008. The [Function Discovery](#) API and the [WsdCodeGen](#) tool are supplemental tools that can be used for client, device host, and service development. The WSDAPI interfaces can be used directly to expose advanced functionality.

Major WSDAPI interfaces

The four major WSDAPI interfaces are [IWSDDiscoveryProvider](#), [IWSDDiscoveryPublisher](#), [IWSDDeviceProxy](#), and [IWSDDeviceHost](#). For a list of all of the WSDAPI interfaces, see [Web Services on Devices Interfaces](#).

IWSDDiscoveryProvider

[IWSDDiscoveryProvider](#) is used to implement WS-Discovery functionality on clients.

[IWSDDiscoveryProvider](#) issues WS-Discovery [Probe](#) and [Resolve](#) messages, and receives [Hello](#), [Bye](#), [ProbeMatches](#), and [ResolveMatches](#) messages. Use the information retrieved through the [IWSDDiscoveryProvider](#) interface when creating an [IWSDDeviceProxy](#) interface used to describe and control a specific DPWS device.

An [IWSDDiscoveryProvider](#) interface is not necessary when simply resolving a particular DPWS device address before creating a device proxy. [WSDCreateDeviceProxy](#) will automatically resolve the device address if required.

The [Function Discovery](#) API can be used for generic device and service discovery, as the API can discover DPWS devices and also devices using other protocols. Consider using Function Discovery when writing a generic discovery application.

IWSDDiscoveryPublisher

[IWSDDiscoveryPublisher](#) is used to implement WS-Discovery functionality on target services, such as devices.

[IWSDDiscoveryPublisher](#) allows an application to publish its presence using WS-Discovery Hello and Bye messages. This interface allows an application to receive Probe

and Resolve requests, and construct and send ProbeMatches and ResolveMatches responses.

An [IWSDDiscoveryPublisher](#) interface is not necessary when simply publishing the existence of an [IWSDDeviceHost](#) object. [IWSDDeviceHost](#) manages its own WS-Discovery presence.

IWSDDeviceProxy

[IWSDDeviceProxy](#) is used to implement client-side WS-Discovery, WS-MetadataExchange, and control functionality. This functionality includes optional secure channel, WS-Eventing, and attachment capabilities.

The [IWSDDeviceProxy](#) interface has the following three uses.

- Resolves logical device addresses, if necessary.
- Initiates metadata requests to devices to enumerate the types and addresses of services.
- Provides a source for [IWSDServiceProxy](#) objects, which can be used to issue control messages to specific services on a device.

The [IWSDDeviceProxy](#) object is typically created and used entirely inside code generated by [WsdCodeGen](#).

IWSDDeviceHost

[IWSDDeviceHost](#) is used to implement device-side WS-Discovery, WS-MetadataExchange, and service hosting functionality. Hosted services may respond to control messages, and may support secure channel, WS-Eventing, and attachment capabilities.

The [IWSDDeviceHost](#) interface has the following uses.

- Hosts service objects.
- Announces the presence of a device host on the network using WS-Discovery.
- Responds to WS-MetadataExchange requests and describes the types and locations of hosted services.
- Dispatches network requests into service objects.

WS-Discovery, WS-MetadataExchange, and WS-Eventing subscription management functionality is handled entirely within the device host object. Before a service can be hosted inside a device host, the following requirements must be met.

- The host must be created by calling [WSDCreateDeviceHost](#).
- The metadata associated with the service must be registered.
- The service itself must be registered.
- The device host must be started.

The [IWSDDeviceHost](#) object is typically created and used inside code generated by [WsdCodeGen](#).

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Using Web Services on Devices

Article • 06/19/2021

The Microsoft Web Services on Devices API (WSDAPI) supports the implementation of client-controlled devices and services, and device hosts conforming to the [Devices Profile for Web Services](#) [↗] (DPWS). WSDAPI may be used for the development of both client and server (device) implementations.

Whenever possible, use WSDAPI to create a Windows-based WSD client or device host application. Using WSDAPI reduces the complexity of development and saves time. For more information, see [WSD Application Development on Windows](#) and [WSDAPI Development Tools](#).

You can also create WSD devices that do not run Windows and are interoperable with Windows-based WSD clients and device hosts. These devices must meet specific requirements to maintain interoperability. For more information, see [WSD Device Development](#).

Related topics

[WSD Application Development on Windows](#)

[WSD Device Development](#)

[WSDAPI Troubleshooting Guide](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD Application Development on Windows

Article • 06/19/2021

The Microsoft Web Services on Devices API (WSDAPI) supports the implementation of client-controlled devices and services, and device hosts conforming to the [Devices Profile for Web Services](#) [↗](#) (DPWS). WSDAPI may be used for the development of both client and server (device) implementations.

Quite often, WSDAPI code for these applications is generated using [WsdCodeGen](#). Some WSDAPI functions and methods are intended to be called only by generated code. The API reference documentation indicates when a function or method should be used or implemented only by generated code.

The Windows SDK includes some sample WSDL files, WsdCodeGen configuration files, and generated code. For more information, see [WSDAPI Samples](#).

If you want to enumerate devices using the WSD protocol and query WSD device metadata, you can use the [Function Discovery](#) API instead.

If you want to implement a WSD device that does not run Windows, see [WSD Device Development](#).

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSDAPI Development Tools

Article • 03/18/2022

The WSDAPI development tools included in the Windows SDK (WSD CodeGen, WSD Debug Host, and WSD Debug Client) enable developers to create and debug WSDAPI-based clients and devices. The source code for these tools is not available. SDK tools are located in the following directory: <Windows SDK Install Folder>\Bin.

WSD CodeGen (wsdcodegen.exe) converts a WSDL contract into generated C++ code, which developers can call directly into. It handles the data serialization and wire representation so developers can focus on designing applications. For more information about WSD CodeGen, see [Web Services on Devices Code Generator](#). This tool is available in the Microsoft Windows Software Development Kit (SDK) and in the Windows Driver Kit (WDK).

The WSD Debug Host (wsddebug_host.exe) and WSD Debug Client (wsddebug_client.exe) tools help developers debug their clients and hosts. These tools can be used to verify and inspect WS-Discovery and metadata exchange traffic. For more information about WSD Debug Host and WSD Debug Client, see [Debugging Tools](#). These tools are available only in the Windows SDK.

There is an additional development tool, named [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) [↗](#), that is available only in the WDK. This tool is used for testing service methods, Message Transmission Optimization Mechanism (MTOM), attachments or WS-Eventing.

Related topics

[WSD Application Development on Windows](#)

[Web Services on Devices Code Generator](#)

[WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) [↗](#)

[Debugging Tools](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Web Services on Devices Code Generator

Article • 03/18/2022

The Web Services on Devices code generator, WsdCodeGen, is a command-line utility that generates program code from a service description. The service description is stored in WSDL and/or XSD files. WsdCodeGen creates C++ and IDL (Interface Definition Language) files. Developers can use WsdCodeGen to create WSDAPI applications without worrying about how the data is marshaled and represented on the wire.

WsdCodeGen can be used to generate code for a service, a client, or both. The utility generates stub code for services, proxy code for clients, and interface and type code for both.

This tool is available in the Microsoft Windows Software Development Kit (SDK) and in the Windows Driver Kit (WDK). SDK tools are located in the following directory:
<Windows SDK Install Folder>\Bin.

The Windows SDK includes sample code generated by WsdCodeGen. For more information, see [WSDAPI Samples](#).

Related topics

[About WsdCodeGen](#)

[Using WsdCodeGen](#)

[WsdCodeGen Reference](#)

[WSD Application Development on Windows](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at [Microsoft Q&A](#)

About WsdCodeGen

Article • 01/07/2021

WsdCodeGen uses an XML configuration file to determine the location of the service metadata. The configuration file is also used to define interface names, interface GUIDs, class names, method names, and other identifiers. For more information about this file, see [WsdCodeGen Configuration File](#).

WsdCodeGen requires two types of input files: an XML configuration file and one or more service description files (WSDL and/or XSD files). WsdCodeGen processes these input files and generates two type of output files: interface files and header/source files.

Input Files

Type	Description
Configuration file	An XML file that indicates the location of the service metadata and defines interface names, interface GUIDs, class names, method names, and other identifiers.
Service description files	One or more WSDL or XSD files that describes the services to implement on the device.

Output Files

Type	Description
Interface files	An IDL (Interface Definition Language) file that can be used with the MIDL compiler to produce an interface header file. WSDAPI clients and WSDAPI services can use this interface file.
C++ header and source files	C++ files that describe the message contract, namespace, and type information. They may contain proxy code and/or stub code. Proxy code implements a service's interface and translates service method calls into WSDAPI operations that make service requests. Stub code translates WSDAPI service requests into code that calls service methods.

Related topics

[Web Services on Devices Code Generator](#)

[Using WsdCodeGen](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

WsdCodeGen Configuration File

Article • 01/07/2021

A WsdCodeGen configuration file is usually generated by the WsdCodeGen tool. You can create configuration files manually, but the complexity and length of the file typically precludes hand-coding. We strongly recommend using WsdCodeGen to generate the file. For more information about generating configuration files, see [Using WsdCodeGen](#) and [WsdCodeGen Command Line Syntax](#).

You should inspect the generated configuration file, and if necessary, modify it before using it to create source code. The configuration file generated by WsdCodeGen is typically sufficient for most client development.

To use the configuration file for server development, some modifications are required. If hosting is enabled (i.e., if "all" or "host" mode is selected), modify the contents of the [ThisModelMetadata](#) element and its child elements as necessary. Also, modify or remove the [PnPXDeviceCategory](#), [PnPXHardwareId](#), and [PnPXCompatibleId](#) elements inside the [ThisModelMetadata](#) element or [Hosted](#) elements as necessary.

A configuration file consists of a sequence of elements that provide input data for code generation followed by any number of [file](#) elements that describe the files to be generated. Input data includes a few global properties and references to types expressed in WSDL, XSD, and managed assemblies. Text and CDATA in [file](#) elements are written to the generated files without modification. Other elements in [file](#) elements are replaced in the generated files with generated code.

XML configuration files must follow a few general rules in order to be properly formatted for use with the code generator utility. These are:

- The root element of any configuration file is [wsdCodeGen](#).
- Elements that contain simple data types are interchangeable with attributes. For example:

syntax

```
<wsdCodeGen>
  <layerNumber>1</layerNumber>
</wsdCodeGen>
```

is equivalent to:

syntax

```
<wsdCodeGen layerNumber="1"/>
```

- In general, there is no constraint on the ordering of elements. For example:

syntax

```
<wsdCodeGen>
  <layerNumber>1</layerNumber>
  <layerPrefix>MEDIA_</layerPrefix>
</wsdCodeGen>
```

is equivalent to:

syntax

```
<wsdCodeGen>
  <layerPrefix>MEDIA_</layerPrefix>
  <layerNumber>1</layerNumber>
</wsdCodeGen>
```

However, the code generator does process the configuration file in a single pass, and ordering does have some relevance. For example, [file](#) elements that generate code relating to a particular port type must occur after the element that instructs the code generator to read the port type contract.

For a complete list of elements used in WsdCodeGen configuration files, see [WsdCodeGen Configuration File XML Reference](#).

Sample configuration files are included with the Windows SDK. For more information, see [WSDAPI Samples](#).

Related topics

[About WsdCodeGen](#)

[WSDAPI Samples](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Using WsdCodeGen

Article • 01/07/2021

To build a WSDAPI application using WsdCodeGen

1. Define the functional contract for the device host in a WSDL file.
2. Generate a configuration file using WsdCodeGen. For more information, see [WsdCodeGen Configuration File](#) and [WsdCodeGen Command Line Syntax](#).
3. Open the generated configuration file and modify the file as required. If you are building a client application, little or no modification is required. If you are building a host application, change the user-specific configuration elements (such as [manufacturer](#) and [modelName](#)).
4. Generate code using WsdCodeGen, providing the configuration file as input. For more information, see [WsdCodeGen Command Line Syntax](#).
5. Use the generated code to build a client, host, or both.

The Windows SDK includes some sample WSDL files, WsdCodeGen configuration files, and generated code. For more information, see [WSDAPI Samples](#).

Related topics

[WSDAPI Samples](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Debugging Tools

Article • 03/18/2022

A debugging toolset built on Web Services on Devices API (WSDAPI) is available in the Windows SDK and the Windows Driver Kit (WDK). These tools can be used to test the functionality of custom applications written on WSDAPI, or devices and clients written using other Device Profile for Web Services (DPWS) stacks.

The WSD Debug Host (wsddebug_host.exe) and WSD Debug Client (wsddebug_client.exe) tools can be used to inspect the characteristics of DPWS clients or hosts. They can also be used to troubleshoot connectivity or configuration problems. For more information, see [WSDAPI Troubleshooting Guide](#). These tools are only available in the SDK. SDK tools are located in the following directory: <Windows SDK Install Folder>\Bin.

The [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) [↗](#) can be used to test SOAP-level or transport-level interoperability (that is, ensuring messages are well-formed). This tool is only available in the WDK.

The WSD Debug Client

The WSD Debug Client (wsddebug_client.exe) provides an interactive console that can be used to send and receive WS-Discovery messages, and to obtain metadata. It can also be used to generate and consume raw multicast messages.

The WSD Debug Client operates in one of three modes: multicast, discovery, and metadata.

Mode	Description
Multicast	In Multicast mode, the WSD Debug Client sends and receives unformatted multicast messages on UDP port 3702, as defined in WS-Discovery. The user may save these SOAP messages in a text file, and may modify and rebroadcast the messages with the WSD Debug Client.
Discovery	In Discovery mode, the WSD Debug Client sends and receives formatted WS-Discovery messages. It can display received Hello , Bye , ProbeMatches , and ResolveMatches messages. It can send Probe messages over UDP or HTTP, and Resolve messages over UDP.
Metadata	In addition to implementing all of the features of Discovery mode, Metadata mode also attempts to retrieve metadata from devices.

For more information, see [Using a Generic Host and Client for HTTP Metadata Exchange](#), [Using a Generic Host and Client for UDP WS-Discovery](#), and [Using WSD Debug Client to Verify Multicast Traffic](#).

The WSD Debug Host

The WSD Debug Host (wsddebug_host.exe) provides an interactive console used to announce the host, respond to client requests, and print diagnostic information.

The WSD Debug Host operates in one of two modes: discovery and metadata.

Mode	Description
Discovery	In Discovery mode, the WSD Debug Host prints formatted WS-Discovery messages. It also sends Hello and Bye messages, and automatically responds to Probe and Resolve messages.
Metadata	In addition to implementing all of the features of Discovery mode, Metadata mode advertises a metadata service and allows clients to connect and perform metadata exchange.

For more information, see [Using a Generic Host and Client for HTTP Metadata Exchange](#) and [Using a Generic Host and Client for UDP WS-Discovery](#).

Related topics

[WSD Application Development on Windows](#)

[WSDAPI Development Tools](#)

[WSDAPI Troubleshooting Guide](#)

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

WSDAPI Samples

Article • 03/18/2022

There are two WSDAPI samples included with the Windows SDK for Windows Server 2008. The source code for the samples can be found in <Windows SDK Install Folder>\Samples\Web\WSDAPI. This version of the SDK is available from the [Download Center](#) . The samples are not available in the Windows Vista SDK.

The stock quote sample (located in <Windows SDK Install Folder>\Samples\Web\WSDAPI\StockQuote) demonstrates a service with basic messaging functionality. The file service sample (located in <Windows SDK Install Folder>\Samples\Web\WSDAPI\FileService) demonstrates a service with advanced functionality, such as asynchronous messaging, attachments, and eventing.

Both samples include the following types of files.

- WSDL files that contain the service descriptions.
- [WsdCodeGen configuration files](#) used to generate WSDAPI code.
- Generated C++ header and source files.
- Client and service implementation files.
- Visual Studio project and solution files.

Both samples implement device hosts ([IWSDDeviceHost](#)), device proxies ([IWSDDeviceProxy](#)), and service proxies ([IWSDServiceProxy](#)). In addition, the file service sample demonstrates the use of asynchronous messaging ([IWSDAsyncCallback](#), [IWSDAsyncResult](#)), attachments ([IWSDInboundAttachment](#), [IWSDOutboundAttachment](#)) and eventing.

The FileServiceContract.vcproj and StockQuoteContract.vcproj files included with the samples call [WsdCodeGen](#) to generate C++ header and source files from the WSDL file specified in the WsdCodeGen configuration file. This means that if the sample WSDL or WsdCodeGen configuration file is changed and the sample project is rebuilt, WsdCodeGen automatically generates new header and source files that reflect the changes. This is the preferred method for building WSDAPI applications. WsdCodeGen is usually called from the command line. Open the relevant *.vcproj file to view the example WsdCodeGen command line calls.

Related topics

[WSD Application Development on Windows](#)

Feedback

Was this page helpful?



Yes



No

Get help at [Microsoft Q&A](#)

WSD Device Development

Article • 01/07/2021

Device manufacturers and other device implementers can create devices that interoperate with Windows-based WSDAPI clients and hosts without using WSDAPI.

In addition to developing the device functionality, device implementers must implement the [Devices Profile for Web Services](#) [↗] (DPWS).

Whenever possible, use WSDAPI to create a Windows-based WSD client or device host application. Using WSDAPI reduces the complexity of development and saves time. For more information, see [WSD Application Development on Windows](#).

Related topics

[WSD Device Implementation Recommendations](#)

[WSDAPI Specification Compliance](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD Device Implementation Recommendations

Article • 01/07/2021

The following topics are of interest to device manufacturers and other device implementers that create devices that interoperate with Windows-based WSDAPI clients and hosts without using WSDAPI. These topics contain recommendations that help reduce the complexity of device code and increase interoperability with WSDAPI.

- [Implementing a Multi-Homed WSD Device](#)
- [Dispatching SOAP Messages](#)
- [Using IPv6 Link-Local Addresses](#)
- [Using Logical and Physical Addresses](#)

Related topics

[WSD Device Development](#)

[WSDAPI Specification Compliance](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Implementing a Multi-Homed WSD Device

Article • 01/07/2021

[WS-Discovery](#) and the [Devices Profile for Web Services](#) (DPWS) do not describe the implementation of multi-homed devices. This topic describes multi-homed device support in WSDAPI, and provides implementation recommendations to client and device developers. In this topic, it is assumed that discovery messages are sent over both IPv4 and IPv6 (if available) with the same message ID and application sequencing information.

Discovery in a multi-homed environment

As mentioned in [Hello](#) and [XAddrs](#) section of [Additional WS-Discovery Functionality](#), WSDAPI never provides XAddrs in a Hello message. That means the same Hello message can be sent on all network interfaces with the same message ID and application sequencing information. This makes it easier for client collision detection to discard multiple Hello messages from the same device when a client and the device share more than one subnet.

Because the [XAddrs](#) are not sent in the [Hello](#) message, client implementations must send a [Resolve](#) message to get the relevant device address. The Resolve should be sent on all client interfaces with the same message ID, and the device should filter duplicate messages as needed. Using the same message ID for the Resolve message allows the device to pick a preferred interface for communicating with clients if necessary.

When sending a [ResolveMatch](#) message, a device should provide [XAddrs](#) that relate to the network interface over which it is unicasting the message. This practice helps to avoid multiple client connection attempts and complicated retry logic.

Metadata exchange in a multi-homed environment

Implementing metadata exchange in a multi-homed environment is more difficult than implementing discovery because of metadata versioning. If a client requests metadata over multiple interfaces, then the client can receive multiple [GetResponse](#) messages over different interfaces. These GetResponse messages can contain different

Relationship metadata sections with the same metadata version. This reduces the value of the metadata version number.

There is an alternative approach, where a single [GetResponse](#) message is sent in response with all addresses for the service. The disadvantage of this method is that private information, such as the topology of indirectly accessible networks, may be disclosed.

On Windows Vista, the metadata provided by WSDAPI contains only addresses that are valid for the interface upon which the metadata request was received.

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Dispatching SOAP Messages

Article • 01/07/2021

There are many ways to handle dispatching received SOAP messages to the appropriate service. The two simplest mechanisms are transport level dispatch, and address and action dispatch.

Transport level dispatch

With transport level dispatch, the underlying HTTP server (such as the [HTTP API](#)) is used to manage routing of requests to the device and its services. The server provides a different URL for each service and for the device and different sinks are registered for each URL. This allows code to be designed such that each service is isolated from the other, either running as separate components within the same process or running as separate processes.

Transport level dispatch has a few advantages. Messages can be dispatched to the appropriate component without first parsing the SOAP envelope or message body. Also, the existing mechanism for routing messages provided by most HTTP server implementations can be reused, which means custom dispatching code is unnecessary. It also isolates the SOAP processing code between services, which provides a level of security, as secure services avoid having messages travel through common code.

Address and action dispatch

Address and action dispatch relies on the SOAP headers to determine the appropriate service to which the message is dispatched. This model may also use additional information, such as reference parameters, to further help dispatching.

This model encourages code reuse throughout a layered messaging stack, as all code up to the SOAP processor is shared by all services. Also, distinct transport addresses for services are not required, which means that UUID addresses can be used for service endpoints. Address and action dispatch also translates more directly to a programming model. Developers can plug services and devices into a single component which manages routing, rather than having to tie into a HTTP layer or creating separate components for each service.

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Using IPv6 Link-Local Addresses

Article • 01/07/2021

IPv6 link-local addressing in SOAP messages provides a unique challenge, as IPv6 link-local addresses require a scope ID to be meaningful, but the scope ID only has meaning for the local machine. All client and device implementations must remove the scope ID before sending an IPv6 link-local address in a SOAP message. Additionally, when an IPv6 link-local address is received in a message, the interface the message was received on must be known so the complete link local address with scope ID can be reconstructed.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Using Logical and Physical Addresses

Article • 01/07/2021

[WS-Discovery](#)  defines logical addressing using URIs based on the `urn:uuid:` format. The purpose of this addressing scheme is to differentiate the identity of the device from its current IP address. This scheme essentially provides the functionality of DNS names without requiring a name server. [Devices Profile for Web Services](#)  (DPWS) recommends that devices use this addressing scheme.

DPWS also recommends that services use physical (also called transport) addresses. This enables clients that do not natively support WS-Discovery addressing mechanisms to communicate with DPWS services. Also, each service can define its addresses, which enables transport level addressing for device implementations that manage service dispatch at a lower layer. Finally, using physical addresses maximizes interoperability.

The disadvantage of physical addressing is that it adds complexity to device implementations, as the current IP or transport address must be tracked and device metadata must be modified accordingly. For this reason, DPWS does not require services to use transport addresses.

If logical addresses are used, then there are some scenarios where the implementation behavior is undefined. The WS-Discovery specification does not describe what it means for a service to reside at a logical address. R1001 of the WS-Discovery specification recommends against using WS-Discovery on hosted services because of the associated network chatter.

It is not recommended that services reside at logical addresses as this reduces interoperability. If an implementation absolutely must reside at a logical address, then the service should use the same logical address as the device. If this adds too much complexity to the dispatch model on the device, then the recommended solution is to use reference parameters to differentiate the services. WSDAPI will correctly send messages to services if they use the same endpoint address as the device.

Feedback

Was this page helpful?

 Yes

 No

Get help at [Microsoft Q&A](#)

WSDAPI Troubleshooting Guide

Article • 01/07/2021

The purpose of this guide is to help users troubleshoot failures encountered when using WSDAPI discovery APIs, when creating a WSDAPI host or device proxy, or when using operating system functions (such as [Function Discovery](#) or the Network Explorer) that rely on WSDAPI. The primary goal is to help troubleshoot when a client and host cannot see each other on the network.

For WSDAPI users, this guide contains information that will help you successfully troubleshoot a device proxy (using [WSDCreateDeviceProxy](#)), a discovery provider (using [WSDCreateDiscoveryProvider](#)), or a discovery publisher (using [WSDCreateDiscoveryPublisher](#)).

This guide assumes that both the client and host can correctly interoperate with WSDAPI in a controlled environment. Accordingly, this guide is not intended to help troubleshoot DPWS stacks that may be generating improper WS messages. For information on testing interoperability with WSDAPI, see the [WSDAPI Basic Interoperability Tool \(WSDBIT\)](#) in the Windows Driver Kit (WDK).

Before you begin troubleshooting your application, you should familiarize yourself with [Discovery and Metadata Exchange Message Patterns](#).

This guide contains the following sections.

- [Getting Started with WSDAPI Troubleshooting](#)
- [WSDAPI Diagnostic Procedures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Getting Started with WSDAPI Troubleshooting

Article • 08/19/2021

This troubleshooting guide contains a set of [diagnostic procedures](#) that can be used to help identify the cause of application problems. Once the cause of the problem has been successfully identified, the suggested solutions in the diagnostic procedure can be applied in order to resolve the problem.

There are two ways to determine the diagnostic procedure to use. One way is to go to the troubleshooting page for the type of client to view a step-by-step list of diagnostic procedures to use to troubleshoot the client. The other way is to go to the troubleshooting quick reference below to view summary tables that show common problems with WSDAPI applications and the procedures to use to diagnose the problems.

Troubleshooting by Type of Client

The following topics show the relevant diagnostic procedures by type of client. These topics also show the message patterns associated with the client type.

- [Troubleshooting WSDAPI Applications Using Directed Discovery](#)
- [Troubleshooting Function Discovery Clients](#)
- [Troubleshooting People Near Me/Meetings Near Me](#)
- [Troubleshooting the Add Printer Wizard](#)
- [Troubleshooting the Network Explorer](#)
- [Troubleshooting the Projector Wizard](#)
- [Troubleshooting Other WSDAPI Applications](#)

Troubleshooting Quick Reference

The following tables show some problems that can prevent WSDAPI clients and hosts from seeing each other on the network and from exchanging device metadata. The tables also show the diagnostic procedures to run and the criteria to use to evaluate whether the application suffers from a particular problem.

Network environment problems

Problem	Diagnostic Procedure	Problem Identification
The firewall blocks Network Discovery traffic.	Inspecting Adapter and Firewall Settings	Enabling the Network Discovery exception on the firewall solves the problem.
Firewall exceptions specific to the application are blocking messages.	Inspecting Adapter and Firewall Settings	Disabling the firewall solves the problem. WF.msc shows application-specific firewall rules.
The device does not respond to UDP requests by sending a ProbeMatches or ResolveMatches message in a timely fashion (less than 4 seconds).	Inspecting Adapter and Firewall Settings	Disabling the firewall solves the problem, and a generic host that responds in less than 4 seconds works successfully.
The security context of the application is incorrect (that is, the client and host do not have adequate permissions on the network).	Using a Generic Host and Client for UDP WS-Discovery or Using a Generic Host and Client for HTTP Metadata Exchange	The device address is not shown in WSD Debug Client output. Running the application as Administrator solves the problem.
An IPSec policy is blocking messages.	Using a Generic Host and Client for UDP WS-Discovery or Using a Generic Host and Client for HTTP Metadata Exchange	The device address is not shown in WSD Debug Client output. The problem is not solved by disabling the firewall. The problem cannot be reproduced on a machine not subject to any IPSec policies.

Discovery traffic problems

Problem	Diagnostic Procedure	Problem identification
Hello , Probe , or Resolve messages are not transmitted on the network because the application does not correctly enumerate the multicast network interfaces.	Using WSD Debug Client to Verify Multicast Traffic	The Hello, Probe, or Resolve messages do not appear in WSD Debug Client output. The packets do not appear on the network. Packets are not generated for the loopback interface or for other interfaces.
Probe messages are not sent by UDP multicast to port 3702 (for applications not using directed discovery).	Inspecting Network Traces for UDP WS-Discovery	Inspection of the message shows that it was sent to the wrong port.

Problem	Diagnostic Procedure	Problem identification
The Probe message does not contain a Types element, or the Types element is empty.	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the message shows that the Types element is not present or empty.
The Types element of a Probe message does not contain the types to which a host will respond.	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the message shows that the Types element contains a malformed or incorrect value.
A ProbeMatches message was not sent unicast to the UDP port from which the Probe was sent.	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the output shows that no ProbeMatches message was sent or that the message was sent to the wrong port. [!Note] For applications using directed discovery, the ProbeMatches must be sent over HTTP or HTTPS in response to the Probe message.
The ProbeMatches message does not contain a RelatesTo element, or the RelatesTo element is empty.	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the message shows that the RelatesTo element is not present or empty.
The value of the RelatesTo element in a ProbeMatches message does not match the value of the MessageId element from the corresponding Probe message.	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the message shows that the RelatesTo element contains a malformed or incorrect value.

Problem	Diagnostic Procedure	Problem identification
The XAddrs element included in a ProbeMatches message does not conform to the XAddr Validation Rules .	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the message shows that the XAddrs are invalid.
Resolve messages are not sent by UDP multicast to port 3702 (for applications not using directed discovery).	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the output shows that the Resolve message was sent to the wrong port.
A ResolveMatches message was not sent unicast to the UDP port from which a Resolve message was sent.	Inspecting Network Traces for UDP WS-Discovery or Inspecting Network Traces for Applications Using Directed Discovery	Inspection of the output shows that no ResolveMatches message was sent or that the message was sent to the wrong port.

Metadata exchange problems

Problem	Diagnostic Procedure	Problem identification
The transport address advertised by the host is wrong.	Using a Generic Host and Client for HTTP Metadata Exchange	Inspection of the XAddrs in the WSD Debug Client output shows that the transport address is wrong or malformed.
A TCP connection could not be established for metadata exchange.	Inspecting Network Traces for HTTP Metadata Exchange	The output from the packet analyzer does not show the following packet exchange: • A TCP SYN packet sent from the client • A TCP SYN/ACK packet sent from the host • A TCP ACK packet sent from the client

Problem	Diagnostic Procedure	Problem identification
The client did not send a valid HTTP GET request.	Inspecting Network Traces for HTTP Metadata Exchange	There is no HTTP GET request in the packet analyzer output, or the request is malformed.
The client did not send a valid WS-Transfer Get message.	Inspecting Network Traces for HTTP Metadata Exchange	There is no WS-Transfer Get message in the packet analyzer output, or the message is malformed.
The host is not listening on the URL path specified in the HTTP GET request.	Inspecting Network Traces for HTTP Metadata Exchange	There is no HTTP response in the packet analyzer output.
The WS-Transfer Get message does not contain a To element, or the To element is empty.	Inspecting Network Traces for HTTP Metadata Exchange	Inspection of the message shows that the To element is not present or empty.
The value of the To element of a WS-Transfer Get message does not match one of the host's endpoint addresses.	Inspecting Network Traces for HTTP Metadata Exchange	Inspection of the message shows that value of the To element does not match one of the endpoint addresses advertised in the host's ProbeMatches or ResolveMatches message.
The host did not send a valid HTTP response header.	Inspecting Network Traces for HTTP Metadata Exchange	There is no HTTP response in the packet analyzer output, or the request is malformed.

Problem	Diagnostic Procedure	Problem identification
The HTTP response header sent by the host indicates that the request cannot be completed.	Inspecting Network Traces for HTTP Metadata Exchange	The response header has a status code other than HTTP/1.1 200.
The host did not send a valid GetResponse message.	Inspecting Network Traces for HTTP Metadata Exchange	There is no GetResponse message in the packet analyzer output, or the message is malformed.
The GetResponse message does not contain a RelatesTo element, or the RelatesTo element is empty.	Inspecting Network Traces for HTTP Metadata Exchange	Inspection of the message shows that the RelatesTo element is not present or empty.
The value of the RelatesTo element in a GetResponse message does not match the value of the MessageId element from the corresponding Get message.	Inspecting Network Traces for HTTP Metadata Exchange	Inspection of the message shows that the RelatesTo element contains a malformed or incorrect value.

Related topics

[WSDAPI Troubleshooting Guide](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Troubleshooting WSDAPI Applications Using Directed Discovery

Article • 01/07/2021

Applications that use directed discovery send [Probe](#) messages over HTTP or HTTPS to discover devices. The corresponding [ProbeMatches](#) messages sent in response are also sent over HTTP or HTTPS. Directed discovery can be initiated by the Add Printer Wizard, a Function Discovery client, or a WSDAPI application. The Probe and ProbeMatches messages are structurally identical to those sent over UDP. The messages are prefixed with the appropriate HTTP or HTTPS headers.

The following diagnostic procedures should be used (in order) to help identify problems with an application using directed discovery.

To troubleshoot a WSDAPI application using directed discovery

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for HTTP metadata exchange.](#)
3. [Use WinHTTP logging to verify Get traffic.](#)
4. [Inspect network traces for an application using directed discovery.](#)

If the source of the problem cannot be identified using the above diagnostic procedures, follow the directions in [Enabling WSDAPI Tracing](#) and contact Microsoft support.

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

[Troubleshooting the Add Printer Wizard](#)

[Troubleshooting WSDAPI Applications](#)

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

Troubleshooting Function Discovery Clients

Article • 01/07/2021

Function Discovery clients:

- Always use UDP WS-Discovery for device discovery
- Always initiate HTTP or HTTPS connections for metadata exchange
- Sometimes use directed discovery
- Sometimes use a secure channel (HTTPS) for metadata exchange

The following list shows the typical sequence of messages sent and received by Function Discovery clients. Not all messages are mandatory.

1. The client sends a [Probe](#) message to discover devices and services. If the client is using directed discovery then this message is sent over HTTP or HTTPS; otherwise, the message is sent by UDP multicast to port 3702.
2. The client receives [ProbeMatches](#) messages from matching devices or services. Directed discovery messages are sent over HTTP or HTTPS; otherwise, these messages are sent by UDP unicast and originate from port 3702.
3. If no XAddrs were included in the ProbeMatches message, then the client will send a [Resolve](#) message by UDP multicast to port 3702.
4. If a [Resolve](#) message was sent, then the client receives a [ResolveMatches](#) message from matching services. This message is sent by UDP unicast from port 3702 to the port where the Resolve message originated.
5. The client sends a [Get](#) message to request metadata from the device or service. This message is sent by HTTP or HTTPS.
6. The client receives a [GetResponse](#) message with the device or service metadata. This message is sent by HTTP or HTTPS.

The following diagnostic procedures should be used (in order) to help identify problems with a Function Discovery client.

To troubleshoot a Function Discovery client

1. If directed discovery is used, [troubleshoot directed discovery](#).
2. [Inspect adapter and firewall settings](#).
3. [Use a generic host and client for UDP WS-Discovery](#).
4. [Use WSD Debug Client to verify multicast traffic](#).
5. [Inspect network traces for UDP WS-Discovery](#).
6. [Use a generic host and client for HTTP metadata exchange](#).

7. [Use WinHTTP logging to verify Get traffic.](#)
8. [Inspect network traces for HTTP metadata exchange.](#)

If the source of the problem cannot be identified using the above diagnostic procedures, follow the directions in [Enabling WSDAPI Tracing](#) and contact Microsoft support.

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Troubleshooting People Near Me/Meetings Near Me

Article • 01/07/2021

People Near Me/Meetings Near Me:

- Always uses UDP WS-Discovery for device discovery
- Does not do any metadata exchange

The following diagnostic procedures should be used (in order) to help identify problems with People Near Me/Meetings Near Me.

To troubleshoot People Near Me/Meetings Near Me

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for UDP WS-Discovery.](#)
3. [Use WSD Debug Client to verify multicast traffic.](#)
4. [Inspect network traces for UDP WS-Discovery.](#)

If the source of the problem cannot be identified using the above diagnostic procedures, follow the directions in [Enabling WSDAPI Tracing](#) and contact Microsoft support.

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

Troubleshooting the Add Printer Wizard

Article • 01/07/2021

The Add Printer Wizard:

- Always uses UDP WS-Discovery for device discovery
- Always initiates a HTTP or HTTPS connection for metadata exchange
- Sometimes uses directed discovery
- Sometimes uses a secure channel (HTTPS) for metadata exchange

The following diagnostic procedures should be used (in order) to help identify problems with the Add Printer Wizard.

To troubleshoot the Add Printer Wizard

1. If directed discovery is used, [troubleshoot directed discovery](#).
2. [Inspect adapter and firewall settings](#).
3. [Use a generic host and client for UDP WS-Discovery](#).
4. [Use WSD Debug Client to verify multicast traffic](#).
5. [Inspect network traces for UDP WS-Discovery](#).
6. [Use a generic host and client for HTTP metadata exchange](#).
7. [Use WinHTTP logging to verify Get traffic](#).
8. [Inspect network traces for HTTP metadata exchange](#).

If the source of the problem cannot be identified using the above diagnostic procedures, follow the directions in [Enabling WSDAPI Tracing](#) and contact Microsoft support.

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

[Troubleshooting Applications Using Directed Discovery](#)

Feedback

Was this page helpful?

 yes

 no

[Get help at Microsoft Q&A](#)

Troubleshooting the Network Explorer

Article • 01/07/2021

The Network Explorer:

- Always uses UDP WS-Discovery for device discovery
- Always initiates a HTTP or HTTPS connection for metadata exchange
- Sometimes uses a secure channel (HTTPS) for metadata exchange

The following diagnostic procedures should be used (in order) to help identify problems with the Network Explorer.

To troubleshoot the Network Explorer

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for UDP WS-Discovery.](#)
3. [Use WSD Debug Client to verify multicast traffic.](#)
4. [Inspect network traces for UDP WS-Discovery.](#)
5. [Use a generic host and client for HTTP metadata exchange.](#)
6. [Use WinHTTP logging to verify Get traffic.](#)
7. [Inspect network traces for HTTP metadata exchange.](#)

The Network Explorer uses [Function Discovery](#) to enumerate network devices. For more troubleshooting information, see [Troubleshooting Function Discovery Clients](#).

If the source of the problem cannot be identified using the above diagnostic procedures, follow the directions in [Enabling WSDAPI Tracing](#) and contact Microsoft support.

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

[Troubleshooting Function Discovery Clients](#)

Feedback



Was this page helpful?

[Get help at Microsoft Q&A](#)

Troubleshooting the Projector Wizard

Article • 01/07/2021

The Projector Wizard:

- Always uses UDP WS-Discovery for device discovery
- Always uses HTTP for metadata exchange
- Sometimes uses directed discovery

The following diagnostic procedures should be used (in order) to help identify problems with the Projector Wizard.

To troubleshoot the Projector Wizard

1. If directed discovery is used, [troubleshoot directed discovery](#).
2. [Inspect adapter and firewall settings](#).
3. [Use a generic host and client for UDP WS-Discovery](#).
4. [Use WSD Debug Client to verify multicast traffic](#).
5. [Inspect network traces for UDP WS-Discovery](#).
6. [Use a generic host and client for HTTP metadata exchange](#).
7. [Use WinHTTP logging to verify Get traffic](#).
8. [Inspect network traces for HTTP metadata exchange](#).

If the source of the problem cannot be identified using the above diagnostic procedures, follow the directions in [Enabling WSDAPI Tracing](#) and contact Microsoft support.

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Troubleshooting Other WSDAPI Applications

Article • 01/07/2021

Applications can directly call into WSDAPI interfaces and functions to perform device discovery and metadata exchange. The message patterns used by these applications vary.

The goal of this troubleshooting guide is to help WSDAPI application developers successfully implement a device proxy. This guide is not intended to help troubleshoot all aspects of WSDAPI. If the device proxy has been successfully created and the client and host can see each other on the network, then this guide cannot address the application's problems. To troubleshoot these application problems, follow the instructions in [Enabling WSDAPI Tracing](#) and contact Microsoft support for further assistance.

Troubleshooting clients calling WSDCreateDeviceProxy

Applications call [WSDCreateDeviceProxy](#) to create and initialize an instance of the [IWSDDeviceProxy](#) interface. This device proxy object can be used to advertise services on a device and also to exchange metadata.

An application calling [WSDCreateDeviceProxy](#) always uses the following messages.

- [Get](#)
- [GetResponse](#)

An application calling [WSDCreateDeviceProxy](#) sometimes uses the following messages.

- [Resolve](#)
- [ResolveMatches](#)

[Resolve](#) and [ResolveMatches](#) messages are generated when a logical device address (that is, a device address of the form urn:uuid:{guid}) is passed to *pszDeviceId*. These messages are not generated when a physical device address is passed to *pszDeviceId*. When [Resolve](#) and [ResolveMatches](#) messages are used, they are sent before the [Get](#) and [GetResponse](#) messages.

The following diagnostic procedures should be used (in order) to help identify problems with an application calling [WSDCreateDeviceProxy](#) with a physical device address.

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for HTTP metadata exchange.](#)
3. [Use WinHTTP logging to verify Get traffic.](#)
4. [Inspect network traces for HTTP metadata exchange.](#)

The following diagnostic procedures should be used (in order) to help identify problems with an application calling [WSDCreateDeviceProxy](#) with a logical device address.

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for UDP WS-Discovery.](#)
3. [Use WSD Debug Client to verify multicast traffic.](#)
4. [Inspect network traces for UDP WS-Discovery.](#)
5. [Use a generic host and client for HTTP metadata exchange.](#)
6. [Use WinHTTP logging to verify Get traffic.](#)
7. [Inspect network traces for HTTP metadata exchange.](#)

Verify that [Resolve](#) and [ResolveMatches](#) messages are generated and meet traffic requirements. It is not necessary to look for [Probe](#) or [ProbeMatches](#) messages in the WSD Debug Client output or in the network traces.

Troubleshooting clients calling WSDCreateDeviceProxyAdvanced

Applications call [WSDCreateDeviceProxyAdvanced](#) to create and initialize an instance of the [IWSDDeviceProxy](#) interface. Unlike [WSDCreateDeviceProxy](#), [WSDCreateDeviceProxyAdvanced](#) has a *pDeviceAddress* parameter that is used to define the device transport address. If this transport address is specified, then logical address resolution is not required and [Resolve](#) and [ResolveMatches](#) messages are not generated.

If *pDeviceAddress* is set to **NULL** and *pszDeviceId* is a logical address, then address resolution is required and [Resolve](#) and [ResolveMatches](#) messages are generated.

The following diagnostic procedures should be used (in order) to help identify problems with an application calling [WSDCreateDeviceProxyAdvanced](#) with a non-**NULL***pDeviceAddress* parameter. These procedures can also be used when *pDeviceAddress* is **NULL** and *pszDeviceId* is a physical address.

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for HTTP metadata exchange.](#)
3. [Use WinHTTP logging to verify Get traffic.](#)
4. [Inspect network traces for HTTP metadata exchange.](#)

The following diagnostic procedures should be used (in order) to help identify problems with an application calling [WSDCreateDeviceProxyAdvanced](#) with *pDeviceAddress* set to **NULL** and with *pszDeviceId* set to a logical address.

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for UDP WS-Discovery.](#)
3. [Use WSD Debug Client to verify multicast traffic.](#)
4. [Inspect network traces for UDP WS-Discovery.](#)
5. [Use a generic host and client for HTTP metadata exchange.](#)
6. [Use WinHTTP logging to verify Get traffic.](#)
7. [Inspect network traces for HTTP metadata exchange.](#)

Verify that [Resolve](#) and [ResolveMatches](#) messages are generated and meet traffic requirements. It is not necessary to look for [Probe](#) or [ProbeMatches](#) messages in the WSD Debug Client output or in the network traces.

Troubleshooting clients using the **IWSDDiscoveryProvider** interface

Applications calling into the [IWSDDiscoveryProvider](#) interface do not perform metadata exchange. This interface is only used for discovery. The message patterns and troubleshooting procedures are different for each method called on the **IWSDDiscoveryProvider** interface.

When an application calls [IWSDDiscoveryProvider::SearchByType](#), a [Probe](#) message is generated. The Probe message is sent by UDP multicast to port 3702. A [ProbeMatches](#) message is generated in response. The ProbeMatches message is sent by UDP unicast and it originates from port 3702.

When an application calls [IWSDDiscoveryProvider::SearchById](#), a [Resolve](#) message is generated. A Resolve message is sent by UDP multicast to port 3702. A [ResolveMatches](#) message is generated in response. The ResolveMatches is sent by UDP unicast and it originates from port 3702.

The following diagnostic procedures should be used (in order) to help identify problems with an application calling [IWSDDiscoveryProvider::SearchByType](#) or [IWSDDiscoveryProvider::SearchById](#). Verify that the messages generated by the called API satisfy the traffic requirements.

1. [Inspect adapter and firewall settings.](#)
2. [Use a generic host and client for UDP WS-Discovery.](#)
3. [Use WSD Debug Client to verify multicast traffic.](#)

4. [Inspect network traces for UDP WS-Discovery.](#)

If an application calls [IWSDDiscoveryProvider::SearchByAddress](#), then it is a directed discovery application. For more troubleshooting information, see [Troubleshooting Applications Using Directed Discovery](#).

Related topics

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSDAPI Diagnostic Procedures

Article • 01/07/2021

For more information about which diagnostic procedures to use in specific scenarios, [Getting Started with WSDAPI Troubleshooting](#).

This troubleshooting guide includes the following types of diagnostic procedures.

UDP and HTTP diagnostic procedures

- [Downloading Netmon and Sample DPWS Filters](#)
- [Enabling WSDAPI Tracing](#)

UDP diagnostic procedures

- [Inspecting Adapter and Firewall Settings](#)
- [Using a Generic Host and Client for UDP WS-Discovery](#)
- [Using WSD Debug Client to Verify Multicast Traffic](#)
- [Inspecting Network Traces for UDP WS-Discovery](#)

HTTP diagnostic procedures

- [Using a Generic Host and Client for HTTP Metadata Exchange](#)
- [Using WinHTTP Logging to Verify Get Traffic](#)
- [Inspecting Network Traces for HTTP Metadata Exchange](#)

Directed discovery diagnostic procedure

- [Inspecting Network Traces for Applications Using Directed Discovery](#)

Related topics

[WSDAPI Troubleshooting Guide](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Capturing WinHTTP Logs

Article • 05/31/2023

Important

This procedure is available only for OS versions prior to Windows 7 or Windows Server 2008 R2.

[WinHTTP](#) logs can be used to help troubleshoot WSDAPI applications. This is helpful when metadata exchange fails or when SSL/TLS negotiation fails.

This procedure shows how to capture WinHTTP logs on the client PC. The WSDAPI-based client application must not be running when logging is enabled. If the client application is running when logging is enabled, the client and/or the PC must be restarted before WS-Discovery and metadata exchange traffic will appear in the WinHTTP logs.

To capture WinHTTP logs

1. Open an elevated command prompt window on the client PC.
2. Run the following command: **netsh winhttp set tracing trace-file-prefix="C:\Temp\dpws" level=verbose format=ansi state=enabled max-trace-file-size=1073741824**

This command enables WinHTTP logging. All log files will be stored in the C:\Temp directory, and the filenames will begin with the dpws prefix. At most 1 GB of log files will be stored.

3. If the process using WinHTTP on the client is already running, restart the computer. For example, if the [Function Discovery](#) APIs are being used, the computer must be restarted. The Function Discovery APIs call WinHTTP from inside a service host, which may have already started when tracing was enabled.
4. Start the WSDAPI-based client application. The application being debugged or the WSD Debug Client can be used.
5. Reproduce the application failure.
6. Terminate the WSDAPI-based client application.
7. If the process using WinHTTP is not terminated with the client application, restart the computer. For example, if the [Function Discovery](#) APIs are being used, the

computer must be restarted.

8. Run the following command: **netsh winhttp set tracing state=disabled**

This command disables WinHTTP logging.

9. Inspect the DPWS logs in C:\Temp and verify that the required requests and messages were sent.

10. If secure channel (HTTPS) communication is being used, check for SSL/TLS failures.

Once WinHTTP logs have been captured, the logs can be examined to look for the cause of a WSDAPI application failure. Note that the text editor used to view these logs must be run as Administrator. For more information, see [Using WinHTTP Logging to Verify Get Traffic](#).

Related topics

- [WinHTTP](#)
- [Using WinHTTP Logging to Verify Get Traffic](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Downloading Netmon and Sample DPWS Filters

Article • 11/09/2023

Microsoft Network Monitor 3 (Netmon) is a packet analyzer used to inspect network traffic. Netmon must be downloaded before the troubleshooting steps given in [Inspecting Network Traces for UDP WS-Discovery](#) and [Inspecting Network Traces for HTTP Metadata Exchange](#) can be followed. After Netmon has been downloaded, DPWS filters can be used to help isolate traffic of interest.

Downloading Netmon

To download Netmon, go to [Microsoft Network Monitor](#), and follow the instructions. For more information about Netmon, see [Information about Network Monitor 3.0](#).

Sample DPWS Filters

Sometimes, WS-Discovery and metadata exchange troubleshooting must take place on a busy network. The sample filters can be used to help limit the Netmon output to traffic of interest. For more information about using Netmon filters, see [Information about Network Monitor 3.0](#).

The following example shows a filter that limits output to all broadcast WS-Discovery traffic.

ⓘ Note

This filter does not capture traffic from stacks that do not respond to multicast WS-Discovery messages that originate at port 3702. If such a stack is being used, then this sample filter must be modified before use.

syntax

```
// All broadcast WS-Discovery traffic
UDP.Port == 3702
```


The following example shows a filter that limits output to HTTP messages sent to WSDAPI stacks on the default port.

ⓘ Note

Services may send relevant HTTP messages from ports other than 5357. If traffic from other ports is of interest, then this sample filter must be modified before use.

syntax

```
// HTTP messages sent to WSDAPI stacks on default port
TCP.Port == 5357
```

The following example shows a filter that limits output to IPv4 multicast traffic.

syntax

```
// All IPv4 multicast traffic
IPv4.DestinationAddress == 239.255.255.250
```

The following example shows a filter that limits output to IPv6 multicast traffic.

syntax

```
// All IPv6 multicast traffic
IPv6.DestinationAddress == FF02::C
```

Some filters can be combined to further limit results. The following example shows a filter that limits output to IPv4 multicast WS-Discovery traffic.

syntax

```
// All IPv4 multicast WS-Discovery traffic
UDP.Port==3702 && IPv4.DestinationAddress=239.255.255.250
```

When these filters are used, relevant traffic may be excluded from the Netmon results. Exclusion can occur when services reside at ports other than the default ports (5357/5358) and when a DPWS stack does not respond to messages using the default port. In these cases, the filters must be modified before use.

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

Enabling WSDAPI Tracing

Article • 01/07/2021

WSDAPI logs contain debugging information that can be used to find the root cause of WSDAPI application failures. When tracing is enabled, logging information is stored in an .etl file in a user-specified location. This .etl file can be sent to Microsoft developer support for root cause analysis. For information about contacting support, go to <https://support.microsoft.com> .

This procedure must be done twice: once on the client, and once on the host.

To enable WSDAPI tracing

1. Using Notepad or another text editor, create a text file with the following text:

syntax

```
"{480217a9-f824-4bd4-bbe8-f371caaf9a0d}" 0xFF 0xFF  
"{649e3596-2620-4d58-a01f-17aefe8185db}" 0xFF 0xFF  
"{96ab095a-9519-4f5c-81ee-c510b0a45463}" 0xFF 0xFF  
"{f9be9c98-10db-4318-bb61-cb0dde08bf7}" 0xFF 0xFF  
"{db1d0418-105a-4c77-9a25-8f96a19716a4}" 0xFF 0xFF  
"{7e2dbfc7-41e8-4987-bca7-76cadfad765f}" 0xFF 0xFF  
"{8b20d3e4-581f-4a27-8109-df01643a7a93}" 0xFF 0xFF  
"{6d04bf88-60a5-4d02-bc5c-94a20ba490ec}" 0xFF 0xFF  
"{75454210-b231-4fea-b2b4-2cc66d7ae8aa}" 0xFF 0xFF  
"{e176aa66-5cc8-4321-9624-f9c1d2b7bf06}" 0xFF 0xFF  
"{836767a6-af31-4938-b4c0-ef86749a9aef}" 0xFF 0xFF
```

2. Save the text file as `C:\temp\traceguids.txt` and then close the file.
3. Open an elevated command prompt window.
4. Run the following command: **logman.exe create trace wsdlog -o c:\temp\wsd**
5. Run the following command: **logman.exe update wsdlog -pf c:\temp\traceguids.txt**
6. Run the following command: **logman.exe start wsdlog**
7. Reproduce the failure by starting the host and client or by pressing F5 in the Network Explorer.

To disable WSDAPI tracing

1. Open an elevated command prompt window.

2. Run the following command: **logman.exe stop wsdlog**

Once the application failure has been captured, the *.etl files can be sent to Microsoft support. These files are located in `C:\temp\wsd_*.etl`.

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

<https://support.microsoft.com> 

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Inspecting Adapter and Firewall Settings

Article • 01/26/2022

A misconfigured firewall can cause WSD applications to fail. This topic provides some troubleshooting procedures to use when WSD clients and hosts cannot see each other on the network. The firewall settings should be inspected before using any other application troubleshooting procedure.

To inspect the adapter and firewall settings

1. Verify that the **Network Discovery** exception is enabled.
2. Check that there are no application-specific firewall rules blocking the application.
3. Explicitly enable the ports used for discovery and metadata exchange.
4. Disable the firewall and retest the application.

⚠ Note

The firewall should be re-enabled after completing this step.

Verifying that the Network Discovery exception is enabled

If any WS-Discovery applications are running, the **Network Discovery** firewall exception must be allowed.

To enable the Network Discovery firewall exception

1. Click **Start**, click **Run**, and then type **firewall.cpl**. This opens the **Windows Firewall Control Panel** applet.
2. Choose **Allow a program through Windows Firewall**.
3. On the **Exceptions** tab, select the **Network Discovery** check box.
4. Click **OK** to close the firewall applet.

Retest the program after making this firewall change. If the program now works successfully, the cause of the problem has been identified and no further troubleshooting steps are necessary. Otherwise, move on to the next step.

Checking for application-specific firewall rules

Advanced configuration of the Windows Firewall can take place in a Microsoft Management Control (MMC) snap-in named **Windows Firewall with Advanced Security**. This snap-in can be used to troubleshoot suspected firewall problems.

Developers can use the [Windows Firewall with Advanced Security](#) APIs to create firewall rules that apply to their WSD applications. Specifically, the [Add](#) method of the [INetFwRules](#) interface can be used to add a new firewall rule. If firewall rules are created incorrectly, clients and hosts may not be able to see each other on the network.

To check for application-specific firewall rules

1. Click **Start**, click **Run**, and then type **wf.msc**.
2. Look for application-specific rules that may be blocking traffic. For more information, see [Windows Firewall with Advanced Security - Diagnostics and Troubleshooting Tools](#).
3. Remove application-specific rules.

If no application-specific rules were found, move on to the next step. If an application-specific rule was found and removed, retest the program after making the firewall change. If the program now works successfully, the cause of the problem has been identified and no further troubleshooting steps are necessary. Otherwise, move on to the next step.

Enabling the ports used for discovery and metadata exchange

WS-Discovery uses the UDP port 3702 for message exchange. In addition, TCP ports 5357 and 5358 are sometimes used for metadata exchange. These ports can be explicitly opened on the firewall using the procedures described in "Open a port in Windows Firewall".

Retest the program after making this firewall change. If the program now works successfully, the cause of the problem has been identified and no further troubleshooting steps are necessary. Otherwise, move on to the next step.

Disabling the firewall

The Windows Firewall can be disabled to help troubleshoot suspected problems. Other applicable firewalls (such as the firewall on a router) can also be disabled for

troubleshooting purposes. For information about enabling and disabling the Windows Firewall, see [Turn Windows Firewall on or off](#).

Retest the application after disabling any applicable firewalls. If the program now works successfully, then the firewall was blocking the traffic. There are a few possible causes of blocked traffic.

- Application-specific exceptions blocked the traffic. Check for application-specific firewall rules as described above.
- The device took too long to respond to UDP requests. The Windows Firewall may block UDP responses that return more than 4 seconds after the initial request was sent. Continue troubleshooting by following the procedures given in [Using a Generic Host and Client for UDP WS-Discovery](#) to see if the problem reproduces with a host that responds in less than 4 seconds.

If the application still fails after the firewall is disabled, then the firewall is not causing the application failure. Re-enable the firewalls and continue troubleshooting by following the procedures given in [Using a Generic Host and Client for UDP WS-Discovery](#).

Firewalls should always be re-enabled after troubleshooting has finished.

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Inspecting Network Traces for Applications Using Directed Discovery

Article • 06/16/2021

Any network packet analyzer that can display raw packets can be used to inspect HTTP metadata exchange requests. Microsoft Network Monitor 3 (Netmon) is recommended. For more information about Netmon, see [Downloading Netmon and Sample DPWS Filters](#).

To inspect network traces for directed discovery

1. Configure the host and client to run across the network (that is, make sure that the host and client will operate on different machines).
2. Install the packet analyzer (Netmon) on either the client or the host.
3. Configure the packet analyzer to capture traffic on the network adapter connecting the host and the client.
4. Reproduce the failure by starting the host and client or by pressing F5 in the Network Explorer.
5. Filter the results to isolate WS-Discovery and metadata exchange traffic. To view sample Netmon filters, see [Downloading Netmon and Sample DPWS Filters](#).

ⓘ Note

This step is optional.

6. Verify that messages sent between client and host meet basic traffic requirements.

Verifying that messages meet traffic requirements

WSDAPI clients and hosts must send messages that conform to the following criteria. For general information about message patterns, see [Discovery and Metadata Exchange Message Patterns](#).

- [Probe](#) messages must be sent by HTTP or HTTPS, usually to port 5357 or 5358.

- The **Types** element of a [Probe](#) message must be present and must not be empty. It must contain the types to which a host will respond.
- A [ProbeMatches](#) message must be sent to the HTTP or HTTPS port from which the [Probe](#) was sent.
- The **RelatesTo** element of a [ProbeMatches](#) message must be present and must not be empty. Its value must match the value of the **MessageId** element from the corresponding [Probe](#) message.
- If an **XAddrs** element was included in the [ProbeMatches](#) message, the supplied transport addresses must be validated. For more information, see [XAddr Validation Rules](#).
- A [ProbeMatches](#) message must be sent within 4 seconds of the corresponding [Probe](#) message. The Windows Firewall may drop a [ProbeMatches](#) message sent more than 4 seconds after a [Probe](#) message.
- If no **XAddrs** element was included in the [ProbeMatches](#) message, and the client or host will send an HTTP message (such as a [Get](#) metadata exchange request or a service message), then the client or host must send a [Resolve](#) message by HTTP or HTTPS. This message is usually sent to port 5357 or 5358.
- If a [Resolve](#) message is sent, then a [ResolveMatches](#) message must be sent to the HTTP or HTTPS port from which the [Resolve](#) message was sent.
- A [ResolveMatches](#) message must be sent within 4 seconds of the corresponding [Resolve](#) message. The Windows Firewall may drop a [ResolveMatches](#) message sent more than 4 seconds after a [Resolve](#) message.

If the messages sent by the program do not conform to these message requirements, the cause of the problem has been successfully identified and no further troubleshooting steps are necessary. Rewrite the program so that it generates conformant messages and retest the program.

If the source of the problem still cannot be identified, contact Microsoft support for assistance. Before contacting support, collect the appropriate log files to help identify the root cause of the problem. For more information, see [Enabling WSDAPI Tracing](#).

Related topics

[Troubleshooting Applications Using Directed Discovery](#)

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

[Downloading Netmon and Sample DPWS Filters](#)

Feedback

Was this page helpful?



Yes



No

Get help at [Microsoft Q&A](#)

Inspecting Network Traces for HTTP Metadata Exchange

Article • 06/16/2021

Any network packet analyzer that can display raw packets can be used to inspect HTTP metadata exchange requests. Microsoft Network Monitor 3 (Netmon) is recommended. For more information about Netmon, see [Downloading Netmon and Sample DPWS Filters](#).

This diagnostic procedure may not be as useful for clients and hosts using a secure channel for communications because the message contents are encrypted.

To inspect network traces for HTTP metadata exchange

1. Configure the host and client to run across the network (that is, make sure that the host and client will operate on different machines).
2. Install the packet analyzer (Netmon) on either the client or the host.
3. Configure the packet analyzer to capture traffic on the network adapter connecting the host and the client.
4. Reproduce the failure by starting the host and client or by pressing F5 in the Network Explorer.
5. Filter the results to isolate WS-Discovery and metadata exchange traffic. To view sample Netmon filters, see [Downloading Netmon and Sample DPWS Filters](#).

ⓘ Note

This step is optional.

6. Verify that messages sent between client and host meet basic traffic requirements.

Verifying that messages meet traffic requirements

WSDAPI clients and hosts must send messages that conform to the following criteria. For general information about message patterns, see [Discovery and Metadata Exchange](#)

Message Patterns.

- Messages must meet the traffic requirements provided in the topic [Inspecting Network Traces for UDP WS-Discovery](#), unless it is absolutely certain that WS-Discovery is not being used for metadata exchange.
- A TCP connection must be established between the client and the first transport address provided in the **XAddrs** element of a [ProbeMatches](#) or [ResolveMatches](#) message. The following list shows a typical packet exchange used to establish a TCP connection.
 - The client sends a TCP SYN packet to the host at a specified port.
 - The host sends a TCP SYN/ACK packet to the client.
 - The client sends a TCP ACK packet to the host at a specified port.

Once the client has sent a TCP ACK packet, the TCP connection is established. Note that this message exchange will not occur if a TCP connection has previously been established.

- The client must send a valid [Get](#) HTTP request and message.
- The host must be listening on the URL path specified in the [Get](#) HTTP request.
- The **To** element of a [Get](#) metadata message must be present and not empty. The value of the **To** element must match one of the host's endpoint addresses. A host's endpoint address is typically advertised in a [ProbeMatches](#) or [ResolveMatches](#) message.
- The host must send a valid HTTP response header. If the initial request was successful, the response header should contain the HTTP/1.1 200 status code.
- The host must send a valid [GetResponse](#) message.
- The **RelatesTo** element of a [GetResponse](#) message must be present and must not be empty. Its value must match the value of the **MessageId** element from the corresponding [Get](#) message.

If the HTTP requests or metadata exchange messages sent by the program do not conform to these traffic requirements, the cause of the problem has been successfully identified and no further troubleshooting steps are necessary. Rewrite the program so that it generates conformant messages and requests and retest the program.

If the source of the problem still cannot be identified, contact Microsoft support for assistance. Before contacting support, collect the appropriate log files to help identify the root cause of the problem. For more information, see [Enabling WSDAPI Tracing](#).

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

[Downloading Netmon and Sample DPWS Filters](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Inspecting Network Traces for UDP WS-Discovery

Article • 01/07/2021

Any network packet analyzer that can display raw packets can be used to inspect UDP WS-Discovery packets. Microsoft Network Monitor 3 (Netmon) is recommended. For more information about Netmon, see [Downloading Netmon and Sample DPWS Filters](#).

To inspect network traces for UDP WS-Discovery

1. Configure the host and client to run across the network (that is, make sure that the host and client will operate on different machines).
2. Install the packet analyzer (Netmon) on either the client or the host.
3. Configure the packet analyzer to capture traffic on the network adapter connecting the host and the client.
4. Reproduce the failure by starting the host and client or by pressing F5 in the Network Explorer.
5. Filter the results to isolate WS-Discovery traffic. To view sample Netmon filters, see [Downloading Netmon and Sample DPWS Filters](#).

ⓘ Note

This step is optional.

6. Verify that messages sent between client and host meet basic traffic requirements.

Verifying that messages meet traffic requirements

WSDAPI clients and hosts must send messages that conform to the following criteria. For general information about message patterns, see [Discovery and Metadata Exchange Message Patterns](#).

- [Probe](#) messages must be sent by UDP multicast to port 3702.

- The **Types** element of a [Probe](#) message must be present and must not be empty. It must contain the types to which a host will respond.
- A [ProbeMatches](#) message must be sent unicast to the UDP port from which the [Probe](#) was sent.
- The **RelatesTo** element of a [ProbeMatches](#) message must be present and must not be empty. Its value must match the value of the **MessageId** element from the corresponding [Probe](#) message.
- If an **XAddrs** element was included in the [ProbeMatches](#) message, the supplied transport addresses must be validated. For more information, see [XAddr Validation Rules](#).
- A [ProbeMatches](#) message must be sent within 4 seconds of the corresponding [Probe](#) message. The Windows Firewall may drop a [ProbeMatches](#) message sent more than 4 seconds after a [Probe](#) message.
- If no **XAddrs** element was included in the [ProbeMatches](#) message, and the client or host will send an HTTP message (such as a [Get](#) metadata exchange request or a service message), then the client or host must send a [Resolve](#) message by UDP multicast to port 3702.
- If a [Resolve](#) message is sent, then a [ResolveMatches](#) message must be sent unicast to the UDP port from which the [Resolve](#) message was sent.
- A [ResolveMatches](#) message must be sent within 4 seconds of the corresponding [Resolve](#) message. The Windows Firewall may drop a [ResolveMatches](#) message sent more than 4 seconds after a [Resolve](#) message.

If the messages sent by the program do not conform to these message requirements, the cause of the problem has been successfully identified and no further troubleshooting steps are necessary. Rewrite the program so that it generates conformant messages and retest the program.

If the source of the problem still cannot be identified, contact Microsoft support for assistance. Before contacting support, collect the appropriate log files to help identify the root cause of the problem. For more information, see [Enabling WSDAPI Tracing](#).

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

[Downloading Netmon and Sample DPWS Filters](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Using a Generic Host and Client for HTTP Metadata Exchange

Article • 08/25/2021

If the client and host cannot exchange metadata, then a generic host and client can be substituted for the custom host and client to help troubleshoot the issue. If the device address or device metadata does not appear in the WSD Debug Client output, then the supplied transport addresses or the network environment may be causing the failure. For more information about the generic host and client, see [Debugging Tools](#).

If it has been verified that a generic host and client can complete both WS-Discovery and HTTP metadata exchange, then this diagnostic procedure can be skipped and troubleshooting can be continued by following the procedures in [Using WinHTTP Logging to Verify Get Traffic](#).

If either the host or client is an application running on a PC, the generic host or client should be run in the same security context as the actual host or client. For example, if the actual host or client runs as Administrator, then the generic host or client must run as Administrator. Also, if either the host or client is a standalone device, it should be completely replaced by a PC running a generic host or client in a security context that guarantees unlimited network access (for example, running as Administrator).

To using a generic host and client to troubleshoot HTTP metadata exchange

1. Open a command prompt window.
2. Run the following command: **WSDDebug_host.exe /mode metadata /start**

Note

A **Windows Security Alert** dialog box may appear. If so, click **Unblock** to allow the WSD Debug Host to run.

This command generates output similar to the following. Make a note of the device ID.

```
syntax
```

```
WSDAPI Debug Host
```

```
Copyright (C) Microsoft Corporation 2007. All rights reserved.
```

```
Device ID is urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
Host metadata>
```

3. Run the following command: **WSDDebug_client.exe /mode metadata /hello off /resolve <id>**. Replace <id> with the device ID identified in step 2.

⚠ Note

A **Windows Security Alert** dialog box may appear. If so, click **Unblock** to allow the WSD Debug Client to run.

WSD Debug Client generates output similar to the following.

syntax

```
WSDAPI Debug Client
Copyright (C) Microsoft Corporation 2007. All rights reserved.
Client ID is urn:uuid:0f571af7-6b0e-4daf-8054-f2233ac27910
Hello mode is disabled
Client metadata>
*****
*
Add at 02/28/07 15:16:51
+ EPR:
  + Address: urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
+ Types:
  (wsdp) https://schemas.xmlsoap.org/ws/2006/02/devprof:Device
+ XAddrs:
  https://[::1]:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366
+ Metadata version: 2
+ Instance ID: 1
+ Probe/Resolve tag: WSDAPI debug_client
+ Remote transport address: [::1]:3702
+ Local transport address: ::1
+ Local interface GUID: 42133cd4-6a70-11db-bbc9-806e6f6e6963
Client metadata>
*****
*
Getting metadata for host at 02/28/07 15:16:51:
+ Endpoint reference:
  + Address: urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
Using xAddr: https://[::1]:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366
Client metadata>
*****
*
Metadata for host:
+ Endpoint reference:
```

```
+ Address:                urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
Metadata section:
+ Dialect:
  https://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice
+ Friendly name:
  [no lang]: Debugging Host
+ Firmware version:  1.0
+ Serial number:     00000000
Metadata section:
+ Dialect:
  https://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel
+ Manufacturer:
  [no lang]: Microsoft Corporation
+ Manufacturer URL:  https://www.microsoft.com/
+ Model names:
  [no lang]: Microsoft Debugging Host
+ Model number:      https://www.microsoft.com/
End of metadata
Client metadata>
```

The WSD Debug Client may generate a lot of output on a network with many DPWS devices. The output can be redirected to a file for easier analysis. Type **log tee** *<filename>* at the WSD Debug Client prompt to redirect output to a file. Output redirection can be stopped by typing **log tee stop** at the WSD Debug Client prompt.

Make a note of the endpoint reference (EPR) address. This EPR address should match the device ID identified in step 2 above. Also, verify that the WSD Debug Client completely printed the metadata for the device. The device metadata begins with `Metadata for host` and ends with `End of metadata`.

If the device ID and device metadata appears correctly in the WSD Debug Client output, then the application failure is likely not related to the supplied transport addresses, the operating system, or the network environment. Replace the generic host and client with the custom host and client, and continue troubleshooting by following the procedures in [Using WinHTTP Logging to Verify Get Traffic](#).

If the device address and device metadata do not appear in the WSD Debug Client output, then the failure could have one or more of the following causes:

- The transport address advertised by the host is incorrect or malformed. WSD Debug Client attempts to get device metadata from the URL provided in the **XAddr** element of a [ProbeMatches](#) or [ResolveMatches](#) message. The URL used for metadata exchange appears in the WSD Debug Client output, prefixed by the phrase `Using xAddr`. The following example shows the XAddr used for metadata exchange in the above WSD Debug Client output.

Using xAddr: [https://\[::1\]:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366](https://[::1]:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366)

If the supplied XAddrs do not conform to the [XAddr validation rules](#), then the WSD Debug Client cannot get the device's metadata.

- The application is running in the wrong security context. Verify that the application is using the correct credentials and that the client and host have sufficient permission to access the network.
- The firewall configuration is wrong. Follow the instructions in [Inspecting Adapter and Firewall Settings](#) to verify that the Windows Firewall settings are correct and that there are no other rules dropping the packets. The client and host can also be copied onto a "pristine" machine (one with a default operating system installation that has never been joined to a domain) in order to attempt to reproduce the failure.
- An IPSec policy is blocking the application. Copy the client and host onto a machine that is not subject to IPSec policies and try to reproduce the failure.

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Using a Generic Host and Client for UDP WS-Discovery

Article • 08/25/2021

If the client and host cannot see each other on the network, then a generic host and client can be substituted for the custom host and client to help troubleshoot the issue. If the device address does not appear in the WSD Debug Client output, then the network environment is probably causing the failure. For more information about the generic host and client, see [Debugging Tools](#).

If either the host or client is an application running on a PC, the generic host or client should be run in the same security context as the actual host or client. For example, if the actual host or client runs as Administrator, then the generic host or client must run as Administrator. Also, if either the host or client is a standalone device, it should be completely replaced by a PC running a generic host or client.

To using a generic host and client to troubleshoot UDP WS-Discovery

1. Open a command prompt window.
2. Run the following command: **WSDDebug_host.exe /mode metadata /start**

ⓘ Note

A **Windows Security Alert** dialog box may appear. If so, click **Unblock** to allow the WSD Debug Host to run.

This command generates output similar to the following. Make a note of the device ID.

syntax

```
WSDAPI Debug Host
Copyright (C) Microsoft Corporation 2007. All rights reserved.
Device ID is urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
Host metadata>
```

3. Run the following command: **WSDDebug_client.exe /mode metadata /hello off /resolve <id>**. Replace <id> with the device ID identified in step 2.

❗ Note

A **Windows Security Alert** dialog box may appear. If so, click **Unblock** to allow the WSD Debug Client to run.

WSD Debug Client generates output similar to the following.

syntax

```
WSDAPI Debug Client
Copyright (C) Microsoft Corporation 2007. All rights reserved.
Client ID is urn:uuid:0f571af7-6b0e-4daf-8054-f2233ac27910
Hello mode is disabled
Client metadata>
*****
*
Add at 02/28/07 15:16:51
+ EPR:
  + Address:          urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
+ Types:
  (wsdp) https://schemas.xmlsoap.org/ws/2006/02/devprof:Device
+ XAddrs:
  https://[::1]:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366
+ Metadata version:   2
+ Instance ID:        1
+ Probe/Resolve tag:  WSDAPI debug_client
+ Remote transport address: [::1]:3702
+ Local transport address:  ::1
+ Local interface GUID:   42133cd4-6a70-11db-bbc9-806e6f6e6963
Client metadata>
```

The WSD Debug Client may generate a lot of output on a network with many DPWS devices. The output can be redirected to a file for easier analysis. Type **log tee <filename>** at the WSD Debug Client prompt to redirect output to a file. Output redirection can be stopped by typing **log tee stop** at the WSD Debug Client prompt.

Make a note of the endpoint reference (EPR) address. This EPR address should match the device ID identified in step 2 above. If this is the case, then the application failure is likely not related to the operating system or the network environment. Replace the generic host and client with the custom host and client, and continue troubleshooting by following the procedures in [Using WSD Debug Client to Verify Multicast Traffic](#).

If the device ID does not match the EPR address, then the application failure is probably related to the operating system or the network environment. The failure could have one or more of the following causes:

- The application is running in the wrong security context. Verify that the application is using the correct credentials and that the client and host have sufficient permission to access the network.
- The firewall configuration is wrong. Follow the instructions in [Inspecting Adapter and Firewall Settings](#) to verify that the Windows Firewall settings are correct and that there are no other rules dropping the packets. The client and host can also be copied onto a "pristine" machine (one with a default operating system installation that has never been joined to a domain) in order to attempt to reproduce the failure.
- An IPSec policy is blocking the application. Copy the client and host onto a machine not subject to IPSec policies and try to reproduce the failure.

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Using WinHTTP Logging to Verify Get Traffic

Article • 01/07/2021

If the generic host and client succeed but the actual host and client still fail, it is possible that the metadata request is not being initiated. [WinHTTP](#) logging can be used to verify that outbound messages are being generated and sent correctly.

WSDAPI-based client applications use [WinHTTP](#) to connect to devices. WSDAPI-based device hosts do not use WinHTTP. Also, some third-party proxies do not use WinHTTP. When troubleshooting a host or proxy that does not use WinHTTP, skip this diagnostic procedure and continue troubleshooting by following the procedures in [Inspecting Network Traces for HTTP Metadata Exchange](#).

[WinHTTP](#) logging does not show all TCP-level traffic. Skip to [Inspecting Network Traces for HTTP Metadata Exchange](#) if traffic besides the HTTP traffic is of interest.

To use WinHTTP logging to verify Get traffic

1. [Capture the WinHTTP logs](#).
2. Start Notepad or another text editor. The text editor must be run as Administrator.
3. Open the WinHTTP log file.
4. Verify that the required HTTP requests and metadata messages were sent.

If a [Get](#) message for the host is found in the WinHTTP logs, then the metadata requests are being sent to WinHTTP successfully. Continue troubleshooting by following the procedures in [Inspecting Network Traces for HTTP Metadata Exchange](#).

If a [Get](#) message cannot be found for the host in the WinHTTP logs, then the metadata request is not being initiated. This can happen when the host publishes invalid XAddrs. Verify that the XAddrs on the host conform to the [XAddr validation rules](#).

Verifying that the required HTTP requests and metadata messages were sent

The following events must occur for successful metadata exchange:

- The WSDAPI client generates an outbound HTTP request. This request is sent to the WSDAPI host.
- The client sends a [Get](#) message to the host.

These events are captured in the WinHTTP logs.

The following WinHTTP log file snippet shows an outbound HTTP request generated by a WSDAPI client.

syntax

```
16:51:47.893 ::*0000004* :: WinHttpSendRequest(0x36aae0, "", 0, 0x0, 0, 658, 0)
16:51:47.893 ::*0000004* :: WinHttpSendRequest() returning TRUE
16:51:47.897 ::*0000004* :: sending data:
16:51:47.897 ::*0000004* :: 226 (0xe2) bytes
16:51:47.897 ::*0000004* :: <<<<----- HTTP stream follows below ----->>>>
16:51:47.897 ::*0000004* :: POST /dbe17c74-3b21-4f52-addc-b84b444f73a0
HTTP/1.1
16:51:47.897 ::*0000004* :: Content-Type: application/soap+xml
16:51:47.897 ::*0000004* :: User-Agent: WSDAPI
16:51:47.897 ::*0000004* :: Host: 192.168.0.1:5357
16:51:47.897 ::*0000004* :: Content-Length: 658
16:51:47.897 ::*0000004* :: Connection: Keep-Alive
16:51:47.897 ::*0000004* :: Cache-Control: no-cache
16:51:47.897 ::*0000004* :: Pragma: no-cache
16:51:47.897 ::*0000004* ::
16:51:47.897 ::*0000004* ::
16:51:47.897 ::*0000004* :: <<<<----- End ----->>>>
```

The following WinHTTP log file snippet shows a [Get](#) message. This message should immediately follow the HTTP request.

syntax

```
16:51:47.898 ::*0000004* :: WinHttpWriteData(0x36aae0, 0x11aa7c4, 658, 0x0)
16:51:47.899 ::*0000004* :: sending data:
16:51:47.899 ::*0000004* :: 658 (0x292) bytes
16:51:47.899 ::*0000004* :: <<<<----- HTTP stream follows below ----->>>>
16:51:47.899 ::*0000004* :: <?xml version="1.0" encoding="utf-8" ?>
16:51:47.899 ::*0000004* :: <soap:Envelope
xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"><soap:Header>
<wsa:To>urn:uuid:dbe17c74-3b21-4f52-addc-b84b444f73a0</wsa:To>
<wsa:Action>https://schemas.xmlsoap.org/ws/2004/09/transfer/Get</wsa:Action>
<wsa:MessageID>urn:uuid:8506ac50-3646-4621-9680-86f484d87909</wsa:MessageID>
<wsa:ReplyTo>
<wsa:Address>https://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address></wsa:ReplyTo><wsa:From><wsa:Address>urn:uuid:b32467b5-e7ee-4ae3-8a8e-f5aa417c23b6</wsa:Address></wsa:From></soap:Header><soap:Body>
</soap:Body></soap:Envelope>
16:51:47.899 ::*0000004* :: <<<<----- End ----->>>>
```

```
----->>>>
16:51:47.899 ::*0000004* :: WinHttpWriteData() returning TRUE
```

The Action element

(`<wsa:Action>https://schemas.xmlsoap.org/ws/2004/09/transfer/Get</wsa:Action>`) identifies the message as a [Get](#) message. Verify that the value of the **To** element (for example, `<wsa:To>urn:uuid:dbe17c74-3b21-4f52-addc-b84b444f73a0</wsa:To>`) matches the device ID advertised by the host in the original UDP WS-Discovery messages. The device ID advertised by the host can be checked by using the WSD Debug Host. For more information, see [Using a Generic Host and Client for UDP WS-Discovery](#).

In addition, the host's response to the metadata request can be found in the client's WinHTTP logs. The host generates a [GetResponse](#) message in response to the client's [Get](#) message.

The following WinHTTP log file snippet shows an inbound [GetResponse](#) message received by a WSDAPI client.

syntax

```
16:51:47.899 ::*0000004* :: WinHttpReceiveResponse(0x36aae0, 0x0)
16:51:47.899 ::*0000004* :: WinHttpReceiveResponse() returning TRUE
16:51:47.899 ::*Session* :: DllMain(0x73fc0000, DLL_THREAD_ATTACH, 0x0)
16:51:47.902 ::*0000004* :: received data:
16:51:47.902 ::*0000004* :: 1024 (0x400) bytes
16:51:47.902 ::*0000004* :: <<<<----- HTTP stream follows below -----
----->>>>
16:51:47.902 ::*0000004* :: HTTP/1.1 200
16:51:47.902 ::*0000004* :: Content-Type: application/soap+xml
16:51:47.902 ::*0000004* :: Server: Microsoft-HTTPAPI/2.0
16:51:47.902 ::*0000004* :: Date: Fri, 15 Jun 2007 23:51:47 GMT
16:51:47.905 ::*0000004* :: Content-Length: 2228
16:51:47.905 ::*0000004* ::
16:51:47.905 ::*0000004* :: <?xml version="1.0" encoding="utf-8" ?>
16:51:47.905 ::*0000004* :: <soap:Envelope
xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="https://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wsx="https://schemas.xmlsoap.org/ws/2004/09/mex"
xmlns:wsp="https://schemas.xmlsoap.org/ws/2006/02/devprof"
xmlns:un0="http://schemas.microsoft.com/windows/pnpx/2005/10"
xmlns:pub="http://schemas.microsoft.com/windows/pub/2005/07"><soap:Header>
<wsa:To>https://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
<wsa:Action>https://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse</wsa:Action>
<wsa:MessageID>urn:uuid:2884cbcc-2848-4c35-9327-5ab5451a8729</wsa:MessageID>
<wsa:RelatesTo>urn:uuid:8506ac50-3646-4621-9680-86f484d87909</wsa:RelatesTo></soap:Header>
<soap:Body><wsx:Metadata>
<wsx:MetadataSection
Dialect="https://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice">
```

```
<wsdp:ThisDevice><wsd
16:51:47.905 ::*0000004* :: <<<<----- End -----
----->>>>
```

The **Action** element

(`<wsa:Action>https://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse</wsa:Action>`) identifies the message as a [GetResponse](#) message. Verify that the value of the **RelatesTo** element of the GetResponse message matches the value of the **MessageID** element of the [Get](#) message. In this example, the value of the **RelatesTo** element (`<wsa:RelatesTo>urn:uuid:8506ac50-3646-4621-9680-86f484d87909</wsa:RelatesTo>`) matches the value of the **MessageID** element of the Get message (`<wsa:MessageID>urn:uuid:8506ac50-3646-4621-9680-86f484d87909</wsa:MessageID>`).

Related topics

[WinHTTP](#)

[Capturing WinHTTP Logs](#)

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Using WSD Debug Client to Verify Multicast Traffic

Article • 03/18/2022

If the generic host and client can see each other on the network but the actual host and client cannot, it is likely that the problem is in the messages sent between the endpoints over the network. For more information about the generic host and client, see [Using a Generic Host and Client for UDP WS-Discovery](#). Because full network traces can be difficult to collect, filter, and read, the WSD Debug Client tool can be used to print the multicast sides of WS-Discovery messages.

The WSD Debug Client in multicast mode can only inspect half of the messages, since the client cannot print unicast messages. If unicast traffic is of interest, skip directly to [Inspecting Network Traces for UDP WS-Discovery](#).

This procedure shows a method that will display all multicast traffic on the network. To display only multicast traffic to and from the device, see the [Filtering WSD Debug Client Results](#) section below.

To use the WSD Debug Client to verify multicast traffic

1. Configure the host and client to run across the network (that is, make sure that the host and client will operate on different machines).
2. Open a command prompt and run the following command: **WSDDebug_client.exe /mode multicast**
3. Reproduce the failure by starting the host and client or by pressing F5 in the Network Explorer.
4. Verify that messages are being multicast.

If the required messages are displayed in the WSD Debug Client output, then the application failure may be in the multicast message content, or in the existence or content of the corresponding unicast response messages. Continue troubleshooting by following the instructions in [Inspecting Network Traces for UDP WS-Discovery](#).

If the required messages are displayed in the WSD Debug Client output, then it is likely that the source of the application problem has been identified. It is likely that the multicast traffic is not being transmitted on the network. This failure can happen when the application does not enumerate multicast adapters correctly. Applications must explicitly send multicast traffic over all network interfaces; otherwise, packets may not be generated for the loopback interface or for other interfaces. To verify that the packets

are not appearing on the network, follow the instructions in [Inspecting Network Traces for UDP WS-Discovery](#) and look for evidence of missing multicast messages.

Verifying that messages are being multicast

Always verify that [Probe](#) messages are being multicast. Optionally, verify that [Hello](#) and [Resolve](#) messages are being multicast. Note that not all applications use Resolve messages. For more information about message patterns used by clients and hosts, see [Discovery and Metadata Exchange Message Patterns](#) and [Getting Started with WSDAPI Troubleshooting](#).

Messages must be triggered in order to be sent as described in step 3 above. The WSD Debug Client displays the raw SOAP message as output. Because all messages printed by WSD Debug Client in multicast mode are received over a multicast socket, the message destination address is not displayed.

The following sample WSD Debug Client output shows a Probe message. The `<wsa:Action>` element identifies the message as a Probe message. Inspect the `<wsa:Action>` field to verify that the received message was a Probe message.

syntax

```
UDP message at 05/08/07 10:06:55 from soap.udp://[127.0.0.1:49334]
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wsd="https://schemas.xmlsoap.org/ws/2005/04/discovery"
xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/diagnostic" >
  <soap:Header>
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</wsa:To>
  </soap:Header>
  <wsa:Action>https://schemas.xmlsoap.org/ws/2005/04/discovery/Probe</wsa:Action>
  <wsa:MessageID>urn:uuid:256ad815-1576-4e59-8efc-4c1e0f15fdd2</wsa:MessageID>
  </soap:Header>
  <soap:Body>
    <wsd:Probe>
      <wsd:Types>wsdp:Device</wsd:Types>
    </wsd:Probe>
  </soap:Body>
</soap:Envelope>
```

The following sample WSD Debug Client output shows a Hello message. The `<wsa:Action>` element identifies the message as a Hello message.

syntax

```

UDP message at 05/08/07 10:10:49 from soap.udp://[::1]:49343]
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="h
ttp://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wsd="https://schemas.xmlso
ap.org/ws/2005/04/discovery"
xmlns:wsdp="https://schemas.xmlsoap.org/ws/2006/02/d
evprof"><soap:Header><wsa:To>urn:schemas-xmlsoap-
org:ws:2005:04:discovery</wsa:T
o>
<wsa:Action>https://schemas.xmlsoap.org/ws/2005/04/discovery/Hello</wsa:Acti
on>
<wsa:MessageID>urn:uuid:8999e29a-b056-4345-9e13-f42dbedab28a</wsa:MessageID>
<wsd
:AppSequence InstanceId="1" SequenceId="urn:uuid:abb0a2a1-6efc-4242-b8e7-
c02484a
6eea2" MessageNumber="1"></wsd:AppSequence></soap:Header><soap:Body>
<wsd:Hello><
wsa:EndpointReference><wsa:Address>urn:uuid:02a76d74-82d0-43e6-ab09-
16f54ab81ac6
</wsa:Address></wsa:EndpointReference><wsd:Types>wsdp:Device</wsd:Types>
<wsd:Met
adataVersion>1</wsd:MetadataVersion></wsd:Hello></soap:Body></soap:Envelope>

```

Filtering WSD Debug Client Results

Filtering the WSD Debug Client results can help identify incident traffic involving the device. Filtering is only necessary on noisy networks.

There are two ways to filter results. The IP addresses to filter can be explicitly identified when starting the WSD Debug Client. Alternatively, the IP addresses can be specified after the client has started. This section describes both methods.

To specify IP addresses to filter when starting WSD Debug Client

1. Configure the host and client to run across the network (that is, make sure that the host and client will operate on different machines).
2. Collect the IP address(es) of the device. If the device has multiple addresses (for example, it has both IPv4 and IPv6 addresses) all addresses must be collected.
3. Open a command prompt and run the following command: **WSDDebug_client.exe /mode multicast /ip add <device IP>**

<device IP> is an IP address. The following list shows some sample formats for this IP address.

- 192.168.0.1

- ::1
- mydevice.contoso.com

The WSD Debug Client automatically resolves hostnames supplied at the command prompt.

To filter results after starting WSD Debug Client

1. Configure the host and client to run across the network (that is, make sure that the host and client will operate on different machines).
2. Collect the IP address(es) of the device. If the device has multiple addresses (for example, it has both IPv4 and IPv6 addresses) all addresses must be collected.
3. Open a command prompt and run the following command: **WSDDebug_client.exe /mode multicast**
4. At the WSD Debug Client command prompt, run the following command: **ip add <device IP>**
5. Repeat step 4 until all device IP addresses have been added.

The following procedure assumes that the WSD Debug Client has been started and filtering by IP address is occurring.

To verify that the correct IP addresses are being filtered

- At the WSD Debug Client command prompt, run the following command: **ip print**

The list of IP addresses being filtered appears.

The following procedure assumes that the WSD Debug Client has been started and filtering by IP address is occurring.

To disable filtering

- At the WSD Debug Client command prompt, run the following command: **ip clear**

All multicast traffic will now be shown in the debug output.

Related topics

[WSDAPI Diagnostic Procedures](#)

[Getting Started with WSDAPI Troubleshooting](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Web Services on Devices Reference

Article • 01/07/2021

The following topics describe the programming interface for Web Services on Devices:

- [Web Services on Devices Constants](#)
- [Web Services on Devices Enumerations](#)
- [Web Services on Devices Functions](#)
- [Web Services on Devices Interfaces](#)
- [Web Services on Devices Structures](#)
- [WsdCodeGen Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Web Services on Devices Constants

Article • 01/07/2021

The following constants are used by Web Services on Devices.

Constant	Description
WSD_DEFAULT_HOSTING_ADDRESS	The default address (in UrlPrefix format) that a WSDAPI host will use to listen for requests on port 5357.
WSD_DEFAULT_SECURE_HOSTING_ADDRESS	The default secure address (in UrlPrefix format) that a WSDAPI host will use to listen for requests on port 5358.
WSD_DEFAULT_EVENTING_ADDRESS	The default address (in UrlPrefix format) that a WSDAPI host will use to listen for events on port 5358 for secure hosts and on port 5357 for other hosts.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	Wsutil.h

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Web Services on Devices Enumerations

Article • 01/07/2021

The Web Services on Devices programming interface defines and uses the following enumerations:

- [WSD_CONFIG_PARAM_TYPE](#)
- [WSD_PROTOCOL_TYPE](#)
- [WSDEventType](#)
- [WSDUdpMessageType](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

WSD_CONFIG_PARAM_TYPE enumeration (wsdbase.h)

Article 01/31/2022

Specifies the kind of data stored in a [WSD_CONFIG_PARAM](#) structure.

Syntax

C++

```
typedef enum __MIDL___MIDL_itf_wsdbase_0000_0000_0001 {
    WSD_CONFIG_MAX_INBOUND_MESSAGE_SIZE = 1,
    WSD_CONFIG_MAX_OUTBOUND_MESSAGE_SIZE = 2,
    WSD_SECURITY_SSL_CERT_FOR_CLIENT_AUTH = 3,
    WSD_SECURITY_SSL_SERVER_CERT_VALIDATION = 4,
    WSD_SECURITY_SSL_CLIENT_CERT_VALIDATION = 5,
    WSD_SECURITY_SSL_NEGOTIATE_CLIENT_CERT = 6,
    WSD_SECURITY_COMPACTSIG_SIGNING_CERT = 7,
    WSD_SECURITY_COMPACTSIG_VALIDATION = 8,
    WSD_CONFIG_HOSTING_ADDRESSES = 9,
    WSD_CONFIG_DEVICE_ADDRESSES = 10,
    WSD_SECURITY_REQUIRE_HTTP_CLIENT_AUTH = 11,
    WSD_SECURITY_REQUIRE_CLIENT_CERT_OR_HTTP_CLIENT_AUTH = 12,
    WSD_SECURITY_USE_HTTP_CLIENT_AUTH = 13
} WSD_CONFIG_PARAM_TYPE;
```

Constants

WSD_CONFIG_MAX_INBOUND_MESSAGE_SIZE

Value: 1

The *pConfigData* member is a pointer to a **DWORD** that specifies the maximum size, in octets, of an inbound message.

The *dwConfigDataSize* member is 4.

WSD_CONFIG_MAX_OUTBOUND_MESSAGE_SIZE

Value: 2

The *pConfigData* member is a pointer to a **DWORD** that specifies the maximum size, in octets, of an outbound message.

The *dwConfigDataSize* member is 4.

WSD_SECURITY_SSL_CERT_FOR_CLIENT_AUTH

Value: 3

Used to pass in the client certificate that WSDAPI will use for client authentication in an SSL connection.

The *pConfigData* member is a pointer to a [CERT_CONTEXT](#) structure that represents the client certificate. The caller needs to have read access to the private key of the certificate.

The *dwConfigDataSize* member is the size of the [CERT_CONTEXT](#) structure.

WSD_SECURITY_SSL_SERVER_CERT_VALIDATION

Value: 4

Used to pass in the SSL server certificate validation information into WSDAPI. When establishing the SSL connection, WSDAPI will accept only a server certificate that matches the criteria specified by the [WSD_SECURITY_CERT_VALIDATION](#) structure.

The *pConfigData* member is a pointer to a [WSD_SECURITY_CERT_VALIDATION](#) structure.

The *dwConfigDataSize* member is the size of the [WSD_SECURITY_CERT_VALIDATION](#) structure.

WSD_SECURITY_SSL_CLIENT_CERT_VALIDATION

Value: 5

Used to pass in the SSL client certificate validation information into WSDAPI. On incoming SSL connections, if a client certificate is available, WSDAPI will reject the connection if the client certificate doesn't match the validation criteria specified by the [WSD_SECURITY_CERT_VALIDATION](#) structure.

The *pConfigData* member is a pointer to a [WSD_SECURITY_CERT_VALIDATION](#) structure.

The *dwConfigDataSize* member is the size of the [WSD_SECURITY_CERT_VALIDATION](#) structure.

WSD_SECURITY_SSL_NEGOTIATE_CLIENT_CERT

Value: 6

Specifies that on incoming SSL connections, WSDAPI will request a client certificate from the SSL client if one is not already made available by the client. If the remote entity cannot provide a client certificate, the connection will be rejected. Note that the SSL record that is created for that port must explicitly allow for client certificate negotiation.

The *pConfigData* member is **NULL**.

The *dwConfigDataSize* member is 0.

WSD_SECURITY_COMPACTSIG_SIGNING_CERT

Value: 7

Used to specify which certificate is to be used by WSDAPI to sign outbound WS_Discovery UDP messages.

The *pConfigData* member is a pointer to a [CERT_CONTEXT](#) structure that represents the signing certificate. The caller needs to have read access to the certificate's private key..

The *dwConfigDataSize* member is the size of the [CERT_CONTEXT](#) structure.

WSD_SECURITY_COMPACTSIG_VALIDATION

Value: 8

This is used to specify the parameters used to verify inbound signed WS_Discovery UDP message.

The *pConfigData* member is a pointer to a [WSD_SECURITY_SIGNATURE_VALIDATION](#) structure.

The *dwConfigDataSize* member is the size of the [WSD_SECURITY_SIGNATURE_VALIDATION](#) structure.

WSD_CONFIG_HOSTING_ADDRESSES

Value: 9

This applies only to the [WSDCreateDeviceHost2](#) function. It is used to specify an array of addresses on which the device host should be hosted on. The equivalent is functionality provided through the *ppHostAddresses* and *dwHostAddressCount* parameters of the [WSDCreateDeviceHostAdvanced](#) function.

The *pConfigData* member is a pointer to a [WSD_CONFIG_ADDRESSES](#) structure. The **addresses** member of this structure points to an array of [IWSDAddress](#) objects, each of which is an address on which the device host will listen on.

The *dwConfigDataSize* member is the size of the [WSD_CONFIG_ADDRESSES](#) structure.

WSD_CONFIG_DEVICE_ADDRESSES

Value: 10

This applies only to the [WSDCreateDeviceProxy2](#) function. It is used to specify an address for the device for which the proxy is created. The equivalent is functionality provided through the *deviceConfig* parameter of the [WSDCreateDeviceProxyAdvanced](#) function.

The *pConfigData* member is a pointer to a [WSD_CONFIG_ADDRESSES](#) structure. The **addresses** member of this structure points to an array of [IWSDAddress](#) objects, each of which is an address of the device to which the proxy is created. Currently only one such address is allowed.

The *dwConfigDataSize* member is the size of the [WSD_CONFIG_ADDRESSES](#) structure.

WSD_SECURITY_REQUIRE_HTTP_CLIENT_AUTH

Value: 11

Indicates a requirement for HTTP Authentication using one of the auth schemes specified through WSD_SECURITY_HTTP_AUTH_SCHEMES. Specific scenarios include:

- When specified during a [WSDCreateDeviceHost](#) operation, DPWS clients will be required to authenticate messages sent to the Hosted Services of the WSDAPI device host using HTTP Authentication.
- If this value is expressed in conjunction with WSD_SECURITY_SSL_NEGOTIATE_CLIENT_CERT, then WSDAPI will require HTTP clients to send a client certificate and utilize HTTP authentication.

WSD_SECURITY_REQUIRE_CLIENT_CERT_OR_HTTP_CLIENT_AUTH

Value: 12

When this value is specified, WSDAPI will request HTTP clients to send a client certificate. If the client cannot provide one, then WSDAPI will require HTTP authentication. If the client can do neither, it will be rejected by WSDAPI. Specific scenarios include:

- When specified during a [WSDCreateDeviceHost](#) operation, this behavior will apply to web service messages from DPWS clients.

Note This parameter cannot be used in conjunction with WSD_SECURITY_SSL_NEGOTIATE_CLIENT_CERT. If it is, WSDAPI will return E_INVALIDARG.

WSD_SECURITY_USE_HTTP_CLIENT_AUTH

Value: 13

If the server requires authentication, WSDAPI will authenticate using HTTP authentication. Specific scenarios include:

- When specified during a [WSDCreateDeviceHost](#) operation, this behavior will apply to web service messages from DPWS clients.
- If this value is expressed in conjunction with WSD_SECURITY_SSL_CERT_FOR_CLIENT_AUTH, WSDAPI will send the client certificate and authenticate using HTTP authentication if either operation is required by server.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	wsdbase.h (include Windows.h)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

WSD_PROTOCOL_TYPE enumeration (wsdtypes.h)

Article02/22/2024

Identifies the type of protocol supported by a port.

Syntax

C++

```
typedef enum _WSD_PROTOCOL_TYPE {
    WSD_PT_NONE = 0x00,
    WSD_PT_UDP = 0x01,
    WSD_PT_HTTP = 0x02,
    WSD_PT_HTTPS = 0x04,
    WSD_PT_ALL = 0xff
} WSD_PROTOCOL_TYPE;
```

Constants

 Expand table

WSD_PT_NONE Value: 0x00 No protocols supported.
WSD_PT_UDP Value: 0x01 The UDP protocol is supported.
WSD_PT_HTTP Value: 0x02 The HTTP protocol is supported.
WSD_PT_HTTPS Value: 0x04 The HTTPS protocol is supported.
WSD_PT_ALL Value: 0xff The UDP, HTTP, and HTTPS protocols are supported.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[WSD_PORT_TYPE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDEventType enumeration (wsdtypes.h)

Article02/22/2024

Identifies the type of event produced by the session layer.

Syntax

C++

```
typedef enum _WSDEventType {
    WSDet_None = 0,
    WSDet_Incoming_Message = 1,
    WSDet_Incoming_Fault = 2,
    WSDet_Transmission_Failure = 3,
    WSDet_Response_Timeout = 4
} WSDEventType;
```

Constants

 Expand table

WSDet_None Value: 0 No events were detected.
WSDet_Incoming_Message Value: 1 An incoming message was detected.
WSDet_Incoming_Fault Value: 2 An incoming message fault was detected.
WSDet_Transmission_Failure Value: 3 A message transmission failure was detected.
WSDet_Response_Timeout Value: 4 A message response timeout was detected.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDUdpMessageType enumeration (wsdbase.h)

Article02/22/2024

Identifies the type of UDP message.

Syntax

C++

```
typedef enum _WSDUdpMessageType {
    ONE_WAY = 0,
    TWO_WAY
} WSDUdpMessageType;
```

Constants

 Expand table

ONE_WAY Value: 0 The message is a one-way UDP message without a corresponding response. Hello and Bye messages are one-way messages.
TWO_WAY The message is a two-way UDP message with a corresponding response. Probe and Resolve messages are two-way messages.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdbase.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Web Services on Devices Functions

Article • 01/07/2021

The Web Services on Devices programming interface defines and uses the following functions:

- [PWSD_SOAP_MESSAGE_HANDLER](#)
- [WSD_STUB_FUNCTION](#)
- [WSDAllocateLinkedMemory](#)
- [WSDAttachLinkedMemory](#)
- [WSDCreateDeviceHost](#)
- [WSDCreateDeviceHostAdvanced](#)
- [WSDCreateDeviceProxy](#)
- [WSDCreateDeviceProxyAdvanced](#)
- [WSDCreateDiscoveryProvider](#)
- [WSDCreateDiscoveryProvider2](#)
- [WSDCreateDiscoveryPublisher](#)
- [WSDCreateDiscoveryPublisher2](#)
- [WSDCreateHttpAddress](#)
- [WSDCreateHttpMessageParameters](#)
- [WSDCreateOutboundAttachment](#)
- [WSDCreateUdpAddress](#)
- [WSDCreateUdpMessageParameters](#)
- [WSDDetachLinkedMemory](#)
- [WSDFreeLinkedMemory](#)
- [WSDGenerateFault](#)
- [WSDGenerateFaultEx](#)
- [WSDGetConfigurationOption](#)
- [WSDSetConfigurationOption](#)
- [WSDUriDecode](#)
- [WSDUriEncode](#)
- [WSDXMLAddChild](#)
- [WSDXMLAddSibling](#)
- [WSDXMLBuildAnyForSingleElement](#)
- [WSDXMLCleanupElement](#)
- [WSDXMLCreateContext](#)
- [WSDXMLGetNameFromBuiltInNamespace](#)
- [WSDXMLGetValueFromAny](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

PWSD_SOAP_MESSAGE_HANDLER callback function (wsdtypes.h)

Article02/22/2024

References a SOAP message handler for incoming messages. This is an internal function pointer, and it should not be used by WSDAPI clients or services.

Syntax

C++

```
PWSD_SOAP_MESSAGE_HANDLER PwsdSoapMessageHandler;

HRESULT PwsdSoapMessageHandler(
    IUnknown *thisUnknown,
    WSD_EVENT *event
)
{...}
```

Parameters

thisUnknown

Pointer to the object calling this function.

event

A [WSD_EVENT](#) structure containing the message to be handled.

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	The method succeeded.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdypes.h (include Wsdapi.h)

WSD_STUB_FUNCTION callback function (wsdtypes.h)

Article 02/22/2024

Describes a stub function used to handle an incoming message. This function should only be implemented in and used by [generated code](#).

Syntax

C++

```
WSD_STUB_FUNCTION WsdStubFunction;  
  
HRESULT WsdStubFunction(  
    IUnknown *server,  
    IWSDServiceMessaging *session,  
    WSD_EVENT *event  
)  
{...}
```

Parameters

server

Pointer to the service object that was registered as a handler for messages of this type. Service objects are registered by calling one of the following methods:

[IWSDDeviceHost::RegisterService](#), [IWSDDeviceHost::AddDynamicService](#), or [IWSDServiceProxy::SubscribeToOperation](#).

session

Pointer to an [IWSDServiceMessaging](#) object used for sending a fault or message response.

event

Pointer to a [WSD_EVENT](#) structure that contains the data for the current request.

Return value

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	The method succeeded.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdtypes.h (include Wsdapi.h)

WSDAllocateLinkedMemory function (wsdutil.h)

Article02/22/2024

Allocates a linked memory block.

Syntax

C++

```
void * WSDAllocateLinkedMemory(  
    void *pParent,  
    size_t cbSize  
);
```

Parameters

pParent

Pointer to the parent memory block.

cbSize

Size of the memory block to be allocated.

Return value

Pointer to the newly allocated memory block.

Remarks

The memory block allocated by **WSDAllocateLinkedMemory** is linked to a parent memory block and is freed when the parent memory block is freed.

If *pParent* is **NULL** the allocated memory block must be explicitly freed by calling [WSDFreeLinkedMemory](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDAttachLinkedMemory function (wsdutil.h)

Article02/22/2024

Attaches a child memory block to a parent memory block. Multiple children can be attached to a parent memory block.

Syntax

C++

```
void WSDAttachLinkedMemory(  
    void *pParent,  
    void *pChild  
);
```

Parameters

`pParent`

Pointer to the parent memory block.

`pChild`

Pointer to the child memory block.

Return value

None

Remarks

The child memory block is automatically freed when the parent memory block is freed. Both the parent and child memory blocks must have been previously allocated by calls to [WSDAllocateLinkedMemory](#).

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateDeviceHost function (wsdhost.h)

Article 02/22/2024

Creates a device host and returns a pointer to the [IWSDDeviceHost](#) interface.

Syntax

C++

```
HRESULT WSDCreateDeviceHost(  
    [in] LPCWSTR      pszLocalId,  
    [in] IWSDXMLContext *pContext,  
    [out] IWSDDeviceHost **ppDeviceHost  
);
```

Parameters

[in] `pszLocalId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. If *pszLocalId* is a logical address, the host will announce the logical address and then convert the address to a physical address when it receives Resolve or Probe messages.

If *pszLocalId* is a physical address (such as URL prefixed by http or https), the host will use the address as the physical address and will host on that address instead of the default one.

For secure communication, *pszLocalId* must be an URL prefixed by https, and the host will use the SSL/TLS protocol on the port specified in the URL. The recommended port is port 5358, as this port is reserved for secure connections with WSDAPI. If no port is specified, then the host will use port 443. The host port must be configured with an SSL server certificate before calling **WSDCreateDeviceHost**. For more information about the configuration of host ports, see [HttpSetServiceConfiguration](#).

Any URL (http or https) must be terminated with a trailing slash. The URL must contain a valid IP address or hostname.

The following list shows some example values for *pszLocalId*. It is not a complete list of valid values.

- `http://192.168.0.1:5357/`
- `http://localhost/`

- `http://myHostname:5357/`
- `https://192.168.0.1:5358/`
- `https://myHostname/`
- `https://myHostname/myDevice/`
- `https://myHostname:5358/`

[in] `pContext`

An [IWSDXMLContext](#) object that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[out] `ppDeviceHost`

Pointer to an [IWSDDeviceHost](#) object that you use to expose the WSD-specific device semantics associated with a server that responds to incoming requests.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
<code>S_OK</code>	Method completed successfully.
<code>E_INVALIDARG</code>	<i>pszLocalId</i> is NULL or the length in characters of <i>pszLocalId</i> exceeds <code>WSD_MAX_TEXT_LENGTH</code> (8192).
<code>E_POINTER</code>	<i>ppDeviceHost</i> is NULL .
<code>E_OUTOFMEMORY</code>	Insufficient memory to complete the operation.

Remarks

The `WSDCreateDeviceHost` function calls the [IWSDDeviceHost::Init](#) method, which initializes an instance of an [IWSDDeviceHost](#) object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

See also

[WSDCreateDeviceHostAdvanced](#)

WSDCreateDeviceHost2 function (wsdhost.h)

Article 10/13/2021

Creates a device host that can support signed messages and returns a pointer to the [IWSDDeviceHost](#) interface.

Syntax

C++

```
HRESULT WSDCreateDeviceHost2(  
    [in] LPCWSTR          pszLocalId,  
    [in] IWSDXMLContext  *pContext,  
    [in] WSD_CONFIG_PARAM *pConfigParams,  
    [in] DWORD            dwConfigParamCount,  
    [out] IWSDDeviceHost **ppDeviceHost  
);
```

Parameters

[in] `pszLocalId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. If *pszLocalId* is a logical address, the host will announce the logical address and then convert the address to a physical address when it receives Resolve or Probe messages.

If *pszLocalId* is a physical address (such as URL prefixed by http or https), the host will use the address as the physical address and will host on that address instead of the default one.

If *pszLocalId* is an HTTPS URL, the recommended port is port 5358, as this port is reserved for secure connections with WSDAPI. If no port is specified, then the host will use port 443. The host port must be configured with an SSL server certificate before calling [WSDCreateDeviceHost](#). For more information about the configuration of host ports, see [HttpSetServiceConfiguration](#).

Any URL (http or https) must be terminated with a trailing slash. The URL must contain a valid IP address or hostname.

The following list shows some example values for *pszLocalId*. It is not a complete list of valid values.

- `http://192.168.0.1:5357/`
- `http://localhost/`
- `http://myHostname:5357/`
- `https://192.168.0.1:5358/`
- `https://myHostname/`
- `https://myHostname/myDevice/`
- `https://myHostname:5358/`

[in] `pContext`

An [IWSDXMLContext](#) object that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[in] `pConfigParams`

An array of [WSD_CONFIG_PARAM](#) structures that contain the parameters for creating the object.

[in] `dwConfigParamCount`

The total number of structures passed in *pConfigParams*.

[out] `ppDeviceHost`

Pointer to an [IWSDDeviceHost](#) object that you use to expose the WSD-specific device semantics associated with a server that responds to incoming requests.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
<code>S_OK</code>	Function completed successfully.
<code>E_INVALIDARG</code>	<i>pszLocalId</i> is NULL or the length in characters of <i>pszLocalId</i> exceeds <code>WSD_MAX_TEXT_LENGTH</code> (8192).
<code>E_POINTER</code>	<i>ppDeviceHost</i> is NULL .
<code>E_OUTOFMEMORY</code>	Insufficient memory to complete the operation.

Remarks

The `WSDCreateDeviceHost2` function calls the `IWSDDeviceHost::Init` method, which initializes an instance of an `IWSDDeviceHost` object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateDeviceHostAdvanced function (wsdhost.h)

Article 07/27/2022

Creates a device host and returns a pointer to the [IWSDDeviceHost](#) interface.

Syntax

C++

```
HRESULT WSDCreateDeviceHostAdvanced(  
    [in] LPCWSTR      pszLocalId,  
    [in] IWSDXMLContext *pContext,  
    [in] IWSDAddress  **ppHostAddresses,  
    [in] DWORD        dwHostAddressCount,  
    [out] IWSDDeviceHost **ppDeviceHost  
);
```

Parameters

[in] `pszLocalId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. If *pszLocalId* is a logical address, the host will announce the logical address and then convert the address to a physical address when it receives Resolve or Probe messages.

If *pszLocalId* is a physical address (such as URL prefixed by http or https), the host will use the address as the physical address and will host on that address instead of the default one.

For secure communication, *pszLocalId* must be an URL prefixed by https, and the host will use the SSL/TLS protocol on the port specified in the URL. The recommended port is port 5358, as this port is reserved for secure connections with WSDAPI. If no port is specified, then the host will use port 443. The host port must be configured with an SSL server certificate before calling **WSDCreateDeviceHostAdvanced**. For more information about the configuration of host ports, see [HttpSetServiceConfiguration](#).

If either *pszLocalId* or the transport address referenced by *ppHostAddresses* is an URL prefixed by https, then both URLs must be identical. If this is not the case, **WSDCreateDeviceHostAdvanced** will return E_INVALIDARG.

Any URL (http or https) must be terminated with a trailing slash. The URL must contain a valid IP address or hostname.

The following list shows some example values for *pszLocalld*. It is not a complete list of valid values.

- `http://192.168.0.1:5357/`
- `http://localhost/`
- `http://myHostname:5357/`
- `https://192.168.0.1:5358/`
- `https://myHostname/`
- `https://myHostname/myDevice/`
- `https://myHostname:5358/`

[in] `pContext`

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[in] `ppHostAddresses`

A single [IWSDAddress](#) interface or [IWSDTransportAddress](#) interface. The objects provide information about specific addresses that the host should listen on.

If *pszLocalld* contains a logical address, the resulting behavior is a mapping between the logical address and a specific set of physical addresses (instead of a mapping between the logical address and a default physical address).

[in] `dwHostAddressCount`

The number of items in the *ppHostAddresses* array. If *ppHostAddresses* is an [IWSDAddress](#) interface, count must be 1.

[out] `ppDeviceHost`

Pointer to the [IWSDDeviceHost](#) interface that you use to expose the WSD-specific device semantics associated with a server that responds to incoming requests.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
<code>S_OK</code>	Method completed successfully.

E_INVALIDARG	<i>pszLocalId</i> is NULL , the length in characters of <i>pszLocalId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or <i>pszLocalId</i> points to an URL prefixed by https and that URL does not match the URL of the transport address referenced by <i>ppHostAddresses</i> .
E_POINTER	<i>ppDeviceHost</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

The **WSDCreateDeviceHostAdvanced** function calls the [IWSDDeviceHost::Init](#) method, which initializes an instance of an [IWSDDeviceHost](#) object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

See also

[WSDCreateDeviceHost](#)

WSDCreateDeviceProxy function (wsdclient.h)

Article 02/22/2024

Creates a device proxy and returns a pointer to the [IWSDDeviceProxy](#) interface.

Syntax

C++

```
HRESULT WSDCreateDeviceProxy(  
    [in] LPCWSTR      pszDeviceId,  
    [in] LPCWSTR      pszLocalId,  
    [in] IWSDXMLContext *pContext,  
    [out] IWSDDeviceProxy **ppDeviceProxy  
);
```

Parameters

[in] `pszDeviceId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. A physical address is a URI prefixed by http or https. If this address is a URI prefixed by https, then the proxy will use the SSL/TLS protocol.

The device address may be prefixed with the @ character. When *pszDeviceId* begins with @, this function does not retrieve the device metadata when creating the device proxy.

[in] `pszLocalId`

The logical or physical address of the client, which is used to identify the proxy and to act as an event sink endpoint. A logical address is of the form `urn:uuid:{guid}`.

If the client uses a secure channel to receive events, then the address is a URI prefixed by https. This URI should specify port 5358, as this port is reserved for secure connections with WSDAPI. The port must be configured with an SSL server certificate before calling [WSDCreateDeviceProxyAdvanced](#). For more information about port configuration, see [HttpSetServiceConfiguration](#).

[in] `pContext`

An [IWSDXMLContext](#) object that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[out] `ppDeviceProxy`

Pointer to an [IWSDDeviceProxy](#) object that you use to represent a remote WSD device for client applications and middleware.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszDeviceId</i> is NULL , <i>pszLocalId</i> is NULL , the length in characters of <i>pszDeviceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or the length in characters of <i>pszLocalId</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_POINTER	<i>ppDeviceProxy</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

The **WSDCreateDeviceProxy** function calls the [IWSDDeviceProxy::Init](#) method, which initializes an instance of an [IWSDDeviceProxy](#) object.

This function will also retrieve the device metadata, unless the *pszDeviceId* parameter begins with the @ character. To retrieve device metadata after the device proxy has been created, call [IWSDDeviceProxy::BeginGetMetadata](#) and [IWSDDeviceProxy::EndGetMetadata](#) on the returned [IWSDDeviceProxy](#) object.

For information about troubleshooting **WSDCreateDeviceProxy** function calls, see [Troubleshooting WSDAPI Applications](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

See also

[Troubleshooting WSDAPI Applications](#)

WSDCreateDeviceProxy2 function (wsdclient.h)

Article 10/13/2021

Creates a device proxy that can support signed messages and returns a pointer to the [IWSDDeviceProxy](#) interface.

Syntax

C++

```
HRESULT WSDCreateDeviceProxy2(  
    [in] LPCWSTR      pszDeviceId,  
    [in] LPCWSTR      pszLocalId,  
    [in] IWSDXMLContext *pContext,  
    [in] WSD_CONFIG_PARAM *pConfigParams,  
    [in] DWORD         dwConfigParamCount,  
    [out] IWSDDeviceProxy **ppDeviceProxy  
);
```

Parameters

[in] `pszDeviceId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. A physical address is a URI prefixed by http or https. If this address is a URI prefixed by https, then the proxy will use the SSL/TLS protocol.

The device address may be prefixed with the @ character. When *pszDeviceId* begins with @, this function does not retrieve the device metadata when creating the device proxy.

[in] `pszLocalId`

The logical or physical address of the client, which is used to identify the proxy and to act as an event sink endpoint. A logical address is of the form `urn:uuid:{guid}`.

If the client uses a secure channel to receive events, then the address is a URI prefixed by https. This URI should specify port 5358, as this port is reserved for secure connections with WSDAPI. The port must be configured with an SSL server certificate before calling [WSDCreateDeviceProxyAdvanced](#). For more information about port configuration, see [HttpSetServiceConfiguration](#).

[in] pContext

An [IWSDXMLContext](#) object that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[in] pConfigParams

An array of [WSD_CONFIG_PARAM](#) structures that contain the parameters for creating the object.

[in] dwConfigParamCount

The total number of structures passed in *pConfigParams*.

[out] ppDeviceProxy

Pointer to an [IWSDDeviceProxy](#) object that you use to represent a remote WSD device for client applications and middleware.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Function completed successfully.
E_INVALIDARG	<i>pszDeviceId</i> is NULL , <i>pszLocalId</i> is NULL , the length in characters of <i>pszDeviceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or the length in characters of <i>pszLocalId</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_POINTER	<i>ppDeviceProxy</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

The `WSDCreateDeviceProxy2` function calls the [IWSDDeviceProxy::Init](#) method, which initializes an instance of an [IWSDDeviceProxy](#) object.

This function will also retrieve the device metadata, unless the *pszDeviceId* parameter begins with the @ character. To retrieve device metadata after the device proxy has been created, call [IWSDDeviceProxy::BeginGetMetadata](#) and [IWSDDeviceProxy::EndGetMetadata](#) on the returned [IWSDDeviceProxy](#) object.

For information about troubleshooting **WSDCreateDeviceProxy2** function calls, see [Troubleshooting WSDAPI Applications](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateDeviceProxyAdvanced function (wsdclient.h)

Article 07/27/2022

Creates a device proxy and returns a pointer to the [IWSDDeviceProxy](#) interface.

Syntax

C++

```
HRESULT WSDCreateDeviceProxyAdvanced(  
    [in] LPCWSTR      pszDeviceId,  
        IWSDAddress    *pDeviceAddress,  
    [in] LPCWSTR      pszLocalId,  
    [in] IWSDXMLContext *pContext,  
    [out] IWSDDeviceProxy **ppDeviceProxy  
);
```

Parameters

[in] `pszDeviceId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. A physical address is a URI prefixed by http or https. If this address is a URI prefixed by https, then the proxy will use the SSL/TLS protocol.

If either *pszDeviceId* or the *pszLocalId* is an URL prefixed by https, then both URLs must be identical. If this is not the case, **WSDCreateDeviceProxyAdvanced** will return `E_INVALIDARG`.

The device address may be prefixed with the @ character. When *pszDeviceId* begins with @, this function does not retrieve the device metadata when creating the device proxy.

`pDeviceAddress`

An [IWSDAddress](#) interface that defines the device transport address. When *pDeviceAddress* is specified, a device proxy can be created without requiring the resolution of a logical address passed to *pszDeviceId*. This parameter may be **NULL**.

[in] `pszLocalId`

The logical or physical address of the client, which is used to identify the proxy and to act as an event sink endpoint. A logical address is of the form `urn:uuid:{guid}`.

If the client uses a secure channel to receive events, then the address is a URI prefixed by https. This URI should specify port 5358, as this port is reserved for secure connections with WSDAPI. The port must be configured with an SSL server certificate before calling **WSDCreateDeviceProxyAdvanced**. For more information about port configuration, see [HttpSetServiceConfiguration](#).

[in] pContext

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[out] ppDeviceProxy

Pointer to the [IWSDDeviceProxy](#) interface that you use to represent a remote WSD device for client applications and middleware.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszDeviceId</i> is NULL , <i>pszLocalId</i> is NULL , the length in characters of <i>pszDeviceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), the length in characters of <i>pszLocalId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or <i>pszDeviceId</i> points to a URI prefixed by https and that URL does not match the URI passed to <i>pszLocalId</i> .
E_POINTER	<i>ppDeviceProxy</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

The **WSDCreateDeviceProxyAdvanced** function calls the [IWSDDeviceProxy::Init](#) method, which initializes an instance of an [IWSDDeviceProxy](#) object.

This function will also retrieve the device metadata, unless the *pszDeviceId* parameter begins with the @ character. To retrieve device metadata after the device proxy has been created, call

[IWSDDeviceProxy::BeginGetMetadata](#) and [IWSDDeviceProxy::EndGetMetadata](#) on the returned [IWSDDeviceProxy](#) object.

For information about troubleshooting **WSDCreateDeviceProxyAdvanced** function calls, see [Troubleshooting WSDAPI Applications](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

See also

[Troubleshooting WSDAPI Applications](#)

WSDCreateDiscoveryProvider function (wsddisco.h)

Article02/22/2024

Creates an [IWSDDiscoveryProvider](#) object.

Syntax

C++

```
HRESULT WSDCreateDiscoveryProvider(  
    [in] IWSDXMLContext      *pContext,  
    [out] IWSDDiscoveryProvider **ppProvider  
);
```

Parameters

[in] pContext

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[out] ppProvider

Returns a reference to the initialized [IWSDDiscoveryProvider](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateDiscoveryProvider2 function (wsddisco.h)

Article 02/22/2024

Creates an [IWSDDiscoveryProvider](#) object that supports signed messages.

Syntax

C++

```
HRESULT WSDCreateDiscoveryProvider2(  
    [in] IWSDXMLContext      *pContext,  
    [in] WSD_CONFIG_PARAM    *pConfigParams,  
    [in] DWORD                dwConfigParamCount,  
    [out] IWSDDiscoveryProvider **ppProvider  
);
```

Parameters

[in] `pContext`

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[in] `pConfigParams`

An array of [WSD_CONFIG_PARAM](#) structures that contain the parameters for creating the object.

[in] `dwConfigParamCount`

The total number of structures passed in *pConfigParams*.

[out] `ppProvider`

Returns a reference to the initialized [IWSDDiscoveryProvider](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Function completed successfully.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateDiscoveryPublisher function (wsddisco.h)

Article02/22/2024

Creates an [IWSDDiscoveryPublisher](#) object.

Syntax

```
C++

HRESULT WSDCreateDiscoveryPublisher(
    [in]  IWSDXMLContext      *pContext,
    [out] IWSDDiscoveryPublisher **ppPublisher
);
```

Parameters

[in] pContext

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[out] ppPublisher

Returns a reference to the initialized [IWSDDiscoveryPublisher](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppPublisher</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateDiscoveryPublisher2 function (wsddisco.h)

Article 02/22/2024

Creates an [IWSDDiscoveryPublisher](#) object that supports signed messages.

Syntax

C++

```
HRESULT WSDCreateDiscoveryPublisher2(  
    [in] IWSDXMLContext      *pContext,  
    [in] WSD_CONFIG_PARAM    *pConfigParams,  
    [in] DWORD                dwConfigParamCount,  
    [out] IWSDDiscoveryPublisher **ppPublisher  
);
```

Parameters

[in] `pContext`

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[in] `pConfigParams`

An array of [WSD_CONFIG_PARAM](#) structures that contain the parameters for creating the object.

[in] `dwConfigParamCount`

The total number of structures passed in *pConfigParams*.

[out] `ppPublisher`

Returns a reference to the initialized [IWSDDiscoveryPublisher](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Function completed successfully.
E_POINTER	<i>ppPublisher</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateHttpAddress function (wsdbase.h)

Creates an [IWSDHttpAddress](#) object.

Syntax

C++

```
HRESULT WSDCreateHttpAddress(  
    IWSDHttpAddress **ppAddress  
);
```

Parameters

`ppAddress`

Returns a reference to the initialized [IWSDHttpAddress](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppAddress</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
Library	wsdapi.lib
DLL	Wsdapi.dll

Last updated on 10/31/2025

WSDCreateHttpMessageParameters function (wsdbase.h)

Creates an [IWSDHttpMessageParameters](#) object.

Syntax

C++

```
HRESULT WSDCreateHttpMessageParameters(  
    IWSDHttpMessageParameters **ppTxParams  
);
```

Parameters

`ppTxParams`

Returns a reference to the initialized [IWSDHttpMessageParameters](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>ppTxParams</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
Library	wsdapi.lib
DLL	Wsdapi.dll

Last updated on 10/31/2025

WSDCreateOutboundAttachment function (wsdattachment.h)

Article02/22/2024

Creates an [IWSDOutboundAttachment](#) object.

Syntax

C++

```
HRESULT WSDCreateOutboundAttachment(  
    [out] IWSDOutboundAttachment **ppAttachment  
);
```

Parameters

[out] ppAttachment

Returns a reference to the initialized [IWSDOutboundAttachment](#) object. Cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>attachmentOut</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDCreateUdpAddress function (wsdbase.h)

Creates an [IWSDUdpAddress](#) object.

Syntax

C++

```
HRESULT WSDCreateUdpAddress(  
    [in] IWSDUdpAddress **ppAddress  
);
```

Parameters

[in] ppAddress

An [IWSDUdpAddress](#) interface pointer. This parameter cannot be **NULL**.

Return value

This function can return one of these values.

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppAddress</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
Library	wsdapi.lib
DLL	Wsdapi.dll

Last updated on 10/31/2025

WSDCreateUdpMessageParameters function (wsdbase.h)

Article02/22/2024

Retrieves a pointer to the [IWSDUdpMessageParameters](#) interface.

Syntax

```
C++

HRESULT WSDCreateUdpMessageParameters(
    [out] IWSDUdpMessageParameters **ppTxParams
);
```

Parameters

[out] ppTxParams

Pointer to the [IWSDUdpMessageParameters](#) interface that you use to specify how often WSD repeats the message transmission.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppTxParams</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDDetachLinkedMemory function (wsdutil.h)

Article02/22/2024

Detaches a child memory block from its parent memory block.

Syntax

C++

```
void WSDDetachLinkedMemory(  
    void *pVoid  
);
```

Parameters

pVoid

Pointer to the memory block to be detached.

Return value

None

Remarks

The child memory block must have been previously allocated by a call to [WSDAllocateLinkedMemory](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDFreeLinkedMemory function (wsdutil.h)

Article02/22/2024

Frees a memory block previously allocated with [WSDAllocateLinkedMemory](#).

Syntax

C++

```
void WSDFreeLinkedMemory(  
    void *pVoid  
);
```

Parameters

pVoid

Pointer to the memory block to be freed.

Return value

None

Remarks

All children of the memory block are automatically freed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDGenerateFault function (wsdutil.h)

Article 07/27/2022

Generates a SOAP fault.

Syntax

C++

```
HRESULT WSDGenerateFault(  
    [in] LPCWSTR      pszCode,  
    [in] LPCWSTR      pszSubCode,  
    [in] LPCWSTR      pszReason,  
    [in] LPCWSTR      pszDetail,  
    [in] IWSDXMLContext *pContext,  
    [out] WSD_SOAP_FAULT **ppFault  
);
```

Parameters

[in] pszCode

A SOAP fault code.

The list of possible fault codes follows. For a description of each fault code, see the [SOAP Version 1.2 specification](#).

VersionMismatch

MustUnderstand

DataEncodingUnknown

Sender

Receiver

[in] pszSubCode

A fault subcode.

[in] pszReason

A human readable explanation of the fault.

[in] pszDetail

Contains application-specific error information pertaining to the fault.

[in] pContext

An [IWSDXMLContext](#) interface that represents the context in which to generate the fault.

[out] ppFault

A [WSD_SOAP_FAULT](#) structure that contains the generated fault. When the calling application is done with this data, *ppFault* must be freed with a call to [WSDFreeLinkedMemory](#).

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszCode</i> , <i>pszReason</i> , or <i>pContext</i> is NULL .
E_POINTER	<i>ppFault</i> is NULL .

Remarks

SOAP faults provide a way to communicate error information on failed SOAP messages. Different Web Services protocols extend faults to provide contextual error information, and in some cases, like in WS-Eventing, faults are an expected part of specific message patterns as the client determines whether or not the device supports specific features.

The following fault subcodes are not implemented by WSDAPI:

- InvalidMessageInformationHeader
- MessageInformationHeaderRequired
- UnsupportedExpirationType
- InvalidMessage

- FilteringNotSupported

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDGenerateFaultEx function (wsdutil.h)

Article 07/27/2022

Generates a SOAP fault.

Syntax

C++

```
HRESULT WSDGenerateFaultEx(  
    [in] WSDXML_NAME          *pCode,  
    [in] WSDXML_NAME          *pSubCode,  
    [in] WSD_LOCALIZED_STRING_LIST *pReasons,  
    [in] LPCWSTR               pszDetail,  
    [out] WSD_SOAP_FAULT       **ppFault  
);
```

Parameters

[in] pCode

A SOAP fault code.

The list of possible fault codes follows. For a description of each fault code, see the

[SOAP Version 1.2 specification](#).

VersionMismatch

MustUnderstand

DataEncodingUnknown

Sender

Receiver

[in] pSubCode

A fault subcode.

[in] pReasons

A [WSD_LOCALIZED_STRING_LIST](#) structure that contains a list of localized reason codes.

[in] pszDetail

Contains application-specific error information pertaining to the fault.

[out] ppFault

A [WSD_SOAP_FAULT](#) structure that contains the generated fault. *ppFault* must be freed with a call to [WSDFreeLinkedMemory](#).

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszCode</i> or <i>pReasons</i> is NULL .
E_POINTER	<i>ppFault</i> is NULL .

Remarks

SOAP faults provide a way to communicate error information on failed SOAP messages. Different Web Services protocols extend faults to provide contextual error information, and in some cases, like in WS-Eventing, faults are an expected part of specific message patterns as the client determines whether or not the device supports specific features.

The following fault subcodes are not implemented by WSDAPI:

- InvalidMessageInformationHeader
- MessageInformationHeaderRequired
- UnsupportedExpirationType
- InvalidMessage
- FilteringNotSupported

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDGetConfigurationOption function (wsdutil.h)

Article02/22/2024

Gets a WSDAPI configuration option.

Syntax

```
C++

HRESULT WSDGetConfigurationOption(
    DWORD dwOption,
    [out] LPVOID pVoid,
    DWORD cbOutBuffer
);
```

Parameters

`dwOption`

The type of configuration data to get.

 Expand table

Value	Meaning
WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE 0x0001	Get the maximum size, in bytes, of an inbound message. This message size is a value between 32768 and 1048576.

`[out] pVoid`

Pointer to the configuration data.

`cbOutBuffer`

The size, in bytes, of the data pointed to by *pVoid*. If *dwOption* is set to WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE, then this parameter should be set to `sizeof(DWORD)`.

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	The <i>dwOption</i> is not set to WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE, <i>cbOutBuffer</i> is 0, or <i>cbOutBuffer</i> is not the correct size for the type of configuration data.
E_POINTER	<i>pVoid</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDSetConfigurationOption function (wsdutil.h)

Article 10/13/2021

Sets a WSDAPI configuration option.

Syntax

C++

```
HRESULT WSDSetConfigurationOption(  
    DWORD dwOption,  
    [in] LPVOID pVoid,  
    DWORD cbInBuffer  
);
```

Parameters

dwOption

The type of configuration data to set.

 Expand table

Value	Meaning
WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE 0x0001	Set the maximum size, in bytes, of an inbound message.

[in] pVoid

Pointer to the configuration data. If *dwOption* is set to WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE, then *pVoid* should point to a DWORD that represents the size of an inbound message. The size of the message is a value between 32768 and 1048576.

cbInBuffer

The size, in bytes, of the data pointed to by *pVoid*. If *dwOption* is set to WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE, this parameter should be set to `sizeof(DWORD)`.

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	The <i>dwOption</i> is not set to WSDAPI_OPTION_MAX_INBOUND_MESSAGE_SIZE, <i>cbInBuffer</i> is 0, <i>cbInBuffer</i> is not the correct size for the type of configuration data, or <i>pVoid</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDUriDecode function (wsdutil.h)

Article02/22/2024

Decodes a URI according to the rules in RFC2396.

Syntax

C++

```
HRESULT WSDUriDecode(  
    [in]          LPCWSTR source,  
    [in]          DWORD  cchSource,  
    [out]         LPWSTR *destOut,  
    [out, optional] DWORD  *cchDestOut  
);
```

Parameters

[in] source

Contains the URI to be decoded.

[in] cchSource

Specifies the length of *source* in characters.

[out] destOut

Pointer to a string that contains the decoded URI. If *destOut* is not **NULL**, the calling application should free the allocated string by calling [WSDFreeLinkedMemory](#).

[out, optional] cchDestOut

Specifies the length of *destOut* in characters.

Return value

This function can return one of these values.

 Expand table

Return code	Description

S_OK	Function completed successfully.
E_INVALIDARG	<i>source</i> is NULL or <i>cchSource</i> is 0.
E_FAIL	The length in characters of <i>source</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_POINTER	<i>destOut</i> is NULL .

Remarks

WSDUriDecode decodes any encoded characters in *source*. These characters are identified by a percent sign (%) followed by two hexadecimal digits. **WSDUriDecode** decodes single-byte components of multi-byte characters and converts them back to wide character representation in *destOut*.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDUriEncode function (wsdutil.h)

Article02/22/2024

Encodes a URI according to URI encoding rules in RFC2396.

Syntax

C++

```
HRESULT WSDUriEncode(  
    [in]          LPCWSTR source,  
    [in]          DWORD  cchSource,  
    [out]         LPWSTR *destOut,  
    [out, optional] DWORD  *cchDestOut  
);
```

Parameters

[in] source

Contains the URI to be encoded.

[in] cchSource

Specifies the length of *source* in characters.

[out] destOut

Pointer to a string that contains the encoded URI. If *destOut* is not **NULL**, the calling application should free the allocated string by calling [WSDFreeLinkedMemory](#).

[out, optional] cchDestOut

Specifies the length of *destOut* in characters.

Return value

This function can return one of these values.

 Expand table

Return code	Description

S_OK	Function completed successfully.
E_INVALIDARG	<i>source</i> is NULL or <i>cchSource</i> is 0.
E_FAIL	The length in characters of <i>source</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_POINTER	<i>destOut</i> is NULL .

Remarks

WSDUriEncode encodes certain characters in *source* into an escaped encoding format of %XY, where X and Y are hexadecimal digits corresponding to the single-byte representation of that character. Wide characters that occupy multiple bytes are first rendered into UTF-8 multi-byte format, and then escaped into encoded characters.

WSDUriEncode does not encode single-byte alphanumeric characters. It does encode percent signs (%) in *source*.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDXMLAddChild function (wsdutil.h)

Article02/22/2024

Adds a child element.

Syntax

C++

```
HRESULT WSDXMLAddChild(  
    WSDXML_ELEMENT *pParent,  
    WSDXML_ELEMENT *pChild  
);
```

Parameters

pParent

Reference to a [WSDXML_ELEMENT](#) structure that contains the parent element.

pChild

Reference to a [WSDXML_ELEMENT](#) structure that contains the child element.

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pParent</i> is NULL , <i>pChild</i> is NULL , <i>pChild</i> already has a parent, or a sibling could not be added to an existing child of <i>pParent</i> .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDXMLAddSibling function (wsdutil.h)

Article02/22/2024

Adds a sibling element.

Syntax

C++

```
HRESULT WSDXMLAddSibling(  
    [in] WSDXML_ELEMENT *pFirst,  
    [in] WSDXML_ELEMENT *pSecond  
);
```

Parameters

[in] pFirst

Reference to a [WSDXML_ELEMENT](#) structure that contains the first sibling.

[in] pSecond

Reference to a [WSDXML_ELEMENT](#) structure that contains the second sibling.

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pFirst</i> or <i>pSecond</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDXMLBuildAnyForSingleElement function (wsdutil.h)

Article02/22/2024

Creates an XML element with a specified name and value. The created element can be used as the child of an XML **any** element.

Syntax

C++

```
HRESULT WSDXMLBuildAnyForSingleElement(  
    [in] WSDXML_NAME      *pElementName,  
    [in] LPCWSTR          pszText,  
    [out] WSDXML_ELEMENT **ppAny  
);
```

Parameters

[in] pElementName

Reference to a [WSDXML_NAME](#) structure that contains the name of the created element.

[in] pszText

The text value of the created element.

[out] ppAny

Reference to a [WSDXML_ELEMENT](#) that contains the created element. *ppAny* must be freed with a call to [WSDFreeLinkedMemory](#).

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	Method completed successfully.

E_INVALIDARG	<i>pElementName</i> is NULL or the length in characters of <i>pszText</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_POINTER	<i>ppAny</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDXMLCleanupElement function (wsdutil.h)

Article02/22/2024

Frees memory associated with an XML element.

Syntax

C++

```
HRESULT WSDXMLCleanupElement(  
    [in] WSDXML_ELEMENT *pAny  
);
```

Parameters

[in] *pAny*

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pAny</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDXMLCreateContext function (wsdxml.h)

Article02/22/2024

Creates a new [IWSDXMLContext](#) object.

Syntax

C++

```
HRESULT WSDXMLCreateContext(  
    [out] IWSDXMLContext **ppContext  
);
```

Parameters

[out] ppContext

Pointer to a newly allocated [IWSDXMLContext](#) object. If the function fails, this parameter can be NULL.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppContext</i> is NULL.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h
Library	WsdApi.lib
DLL	WsdApi.dll

WSDXMLGetNameFromBuiltinNamespace function (wsdxml.h)

Article02/22/2024

Gets a specified name from the built-in namespace.

Syntax

C++

```
HRESULT WSDXMLGetNameFromBuiltinNamespace(  
    LPCWSTR    pszNamespace,  
    LPCWSTR    pszName,  
    WSDXML_NAME **ppName  
);
```

Parameters

pszNamespace

The namespace to match with a built-in namespace.

pszName

The name to match with a built-in name.

ppName

Reference to a [WSDXML_NAME](#) structure that contains the returned built-in name. The memory usage of *ppName* is managed elsewhere. Consequently, the calling application should not attempt to deallocate *ppName*.

Return value

This function can return one of these values.

 Expand table

Return code	Description
S_OK	Method completed successfully.

E_INVALIDARG	<i>pszNamespace</i> is NULL , <i>pszName</i> is NULL , the length in characters of <i>pszNamespace</i> exceeds WSD_MAX_TEXT_LENGTH (8192), the length in characters of <i>pszName</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or there was no matching name in the built-in namespace.
E_POINTER	<i>ppName</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h
Library	Wsdapi.lib
DLL	Wsdapi.dll

WSDXMLGetValueFromAny function (wsdutil.h)

Article 02/22/2024

Retrieves a text value from a specified child element of an XML **any** element.

Syntax

C++

```
HRESULT WSDXMLGetValueFromAny(  
    [in] LPCWSTR      pszNamespace,  
    [in] LPCWSTR      pszName,  
    [in] WSDXML_ELEMENT *pAny,  
    [out] LPCWSTR      *ppszValue  
);
```

Parameters

[in] pszNamespace

The namespace of the element to retrieve.

[in] pszName

The name of the element to retrieve.

[in] pAny

Reference to a [WSDXML_ELEMENT](#) structure that contains the **any** element that is the parent of the element to retrieve.

[out] ppszValue

The text value of the element specified by *pszNamespace* and *pszName*. The memory usage of *ppszValue* is managed elsewhere. Consequently, the calling application should not attempt to deallocate *ppszValue*.

Return value

This function can return one of these values.

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The length in characters of <i>pszNamespace</i> or <i>pszName</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or <i>pAny</i> is NULL .
E_POINTER	<i>ppszValue</i> is NULL .
E_FAIL	The method failed.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdutil.h (include Wsdapi.h)
Library	Wsdapi.lib
DLL	Wsdapi.dll

Web Services on Devices Interfaces

Article • 01/07/2021

The Web Services on Devices programming interface defines and uses the following interfaces:

- [IWSDAddress](#)
- [IWSDAsyncCallback](#)
- [IWSDAsyncResult](#)
- [IWSDAttachment](#)
- [IWSDDeviceHost](#)
- [IWSDDeviceHostNotify](#)
- [IWSDDeviceProxy](#)
- [IWSEndpointProxy](#)
- [IWSEventingStatus](#)
- [IWSDHttpAddress](#)
- [IWSDHttpMessageParameters](#)
- [IWSDInboundAttachment](#)
- [IWSDiscoveredService](#)
- [IWSDiscoveryProvider](#)
- [IWSDiscoveryProviderNotify](#)
- [IWSDiscoveryPublisher](#)
- [IWSDiscoveryPublisherNotify](#)
- [IWSDMessageParameters](#)
- [IWSDMetadataExchange](#)
- [IWSDOutboundAttachment](#)
- [IWSDScopeMatchingRule](#)
- [IWSDServiceMessaging](#)
- [IWSDServiceProxy](#)
- [IWSDServiceProxyEventing](#)
- [IWSDSignatureProperty](#)
- [IWSDSSLClientCertificate](#)
- [IWSDTransportAddress](#)
- [IWSDUdpAddress](#)
- [IWSDUdpMessageParameters](#)
- [IWSDXMLContext](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

IWSDAddress interface (wsdbase.h)

Article02/22/2024

Provides access to the individual components of a transport address.

Inheritance

The **IWSDAddress** interface inherits from the [IUnknown](#) interface. **IWSDAddress** also has these types of members:

Methods

The **IWSDAddress** interface has these methods.

[Expand table](#)

IWSDAddress::Deserialize
Parses the address, validates its component parts and saves them in the object.
IWSDAddress::Serialize
Assembles the component parts of the address into a string.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

IWSDAddress::Deserialize method (wsdbase.h)

Article02/22/2024

Parses the address, validates its component parts and saves them in the object.

Syntax

C++

```
HRESULT Deserialize(  
    [in] LPCWSTR pszBuffer  
);
```

Parameters

[in] pszBuffer

Address to save in the object.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszBuffer</i> is NULL , its length in characters exceeds WSD_MAX_TEXT_LENGTH (8192), or it does not contain an address in a valid format.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAddress](#)

IWSDAddress::Serialize method (wsdbase.h)

Article02/22/2024

Assembles the component parts of the address into a string.

Syntax

```
C++

HRESULT Serialize(
    [out] LPWSTR pszBuffer,
    [in]  DWORD  cchLength,
    [in]  BOOL   fSafe
);
```

Parameters

[out] pszBuffer

Buffer to receive the assembled address.

[in] cchLength

Length of *pszBuffer*, in bytes.

[in] fSafe

If **TRUE**, the resulting string will be network safe. For example, if you used [IWSDTransportAddress](#) to build an IPv6 address, the serialized string will not contain the IPv6 scope identifier. However, if *fSafe* is **FALSE**, then the resulting string will contain the IPv6 scope identifier. For all other [IWSDAddress](#) derived objects, there is no specific meaning for this parameter (other than ensuring that the method generate portable addresses).

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description

S_OK	Method completed successfully.
E_POINTER	<i>pszBuffer</i> is NULL .
E_ABORT	The method could not be completed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAddress](#)

IWSDAsyncCallback interface (wsdclient.h)

Article02/22/2024

Handles callbacks for the completion of an asynchronous operation.

Inheritance

The **IWSDAsyncCallback** interface inherits from the [IUnknown](#) interface. **IWSDAsyncCallback** also has these types of members:

Methods

The **IWSDAsyncCallback** interface has these methods.

 Expand table

IWSDAsyncCallback::AsyncOperationComplete
Indicates that the asynchronous operation has completed.

Remarks

This interface provides an asynchronous calling pattern in support of WSDAPI messaging and eventing, allowing an application to receive callback notification based on the status of an operation.

The [IWSDAsyncResult](#) interface can be used to wait for event notification or to poll for operation completion if asynchronous callbacks are not required.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdclient.h (include Wsdapi.h)

IWSDAsyncCallback::AsyncOperationComplete method (wsdclient.h)

Article02/22/2024

Indicates that the asynchronous operation has completed.

Syntax

C++

```
HRESULT AsyncOperationComplete(  
    [in] IWSDAsyncResult *pAsyncResult,  
    [in] IUnknown        *pAsyncState  
);
```

Parameters

[in] pAsyncResult

Pointer to an [IWSDAsyncResult](#) object that contains the user-defined state information passed to [IWSDAsyncResult::SetCallback](#).

[in] pAsyncState

The state of the asynchronous operation.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

The value returned by [AsyncOperationComplete](#) is ignored.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncCallback](#)

IWSDAsyncResult interface (wsdclient.h)

Article 10/05/2021

Represents an asynchronous operation.

Inheritance

The **IWSDAsyncResult** interface inherits from the [IUnknown](#) interface. **IWSDAsyncResult** also has these types of members:

Methods

The **IWSDAsyncResult** interface has these methods.

 Expand table

IWSDAsyncResult::Abort
Aborts the asynchronous operation.
IWSDAsyncResult::GetAsyncState
Gets the state of the asynchronous operation.
IWSDAsyncResult::GetEndpointProxy
Retrieves the endpoint proxy for the asynchronous operation.
IWSDAsyncResult::GetEvent
Retrieves a WSD_EVENT structure that contains the result of the event.
IWSDAsyncResult::HasCompleted
Indicates whether the operation has completed.
IWSDAsyncResult::SetCallback
Specifies a callback interface to call when the asynchronous operation has completed.
IWSDAsyncResult::SetWaitHandle
Specifies a wait handle to set when the operation completes.

Remarks

The **IWSDAsyncResult** interface can be used to set a wait handle to receive event or message notification or poll for operation completion. It can also retrieve the state of an asynchronous operation and retrieve the results and response body of the event.

The **IWSDAsyncCallback** interface can be used to provide an asynchronous calling pattern in support of WSDAPI messaging and eventing, allowing an application to receive callback notification based on the status of an operation.

A failed asynchronous operation is treated as a completed asynchronous operation. Error or fault information can be retrieved from the **IWSDAsyncCallback** interface using the **IWSDAsyncCallback::AsyncOperationComplete** method.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

IWSDAsyncResult::Abort method (wsdclient.h)

Article02/22/2024

Aborts the asynchronous operation.

Syntax

```
C++  
  
HRESULT Abort();
```

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

Abort waits for all pending callbacks set with [SetCallback](#) to complete before returning to the caller.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAsyncResult::GetAsyncState method (wsdclient.h)

Article02/22/2024

Gets the state of the asynchronous operation.

Syntax

C++

```
HRESULT GetAsyncState(  
    [out] IUnknown **ppAsyncState  
);
```

Parameters

[out] ppAsyncState

User-defined state information.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	The operation completed successfully.
E_POINTER	<i>ppAsyncState</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAsyncResult::GetEndpointProxy method (wsdclient.h)

Article02/22/2024

Retrieves the endpoint proxy for the asynchronous operation.

Syntax

C++

```
HRESULT GetEndpointProxy(  
    [out] IWSEndpointProxy **ppEndpoint  
);
```

Parameters

[out] ppEndpoint

An [IWSEndpointProxy](#) interface that implements an endpoint proxy.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppEndpoint</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAsyncResult::GetEvent method (wsdclient.h)

Article02/22/2024

Retrieves a [WSD_EVENT](#) structure that contains the result of the event.

Syntax

```
C++

HRESULT GetEvent(
    [out] WSD_EVENT *pEvent
);
```

Parameters

[out] `pEvent`

Reference to a [WSD_EVENT](#) structure that provides data about the event.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pEvent</i> is NULL .
E_FAIL	Event is not yet available or the asynchronous operation has not completed.

Remarks

This method should only be called by [generated code](#) and only after the [IWSDAsyncResult](#) object has signaled that the operation has completed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAsyncResult::HasCompleted method (wsdclient.h)

Article02/22/2024

Indicates whether the operation has completed.

Syntax

```
C++  
  
HRESULT HasCompleted();
```

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	The operation completed.
S_FALSE	The operation has not completed.

Remarks

A failed asynchronous operation is treated as a completed asynchronous operation. Error or fault information can be retrieved from the [IWSDAsyncCallback](#) interface using the [IWSDAsyncCallback::AsyncOperationComplete](#) method.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAsyncResult::SetCallback method (wsdclient.h)

Article02/22/2024

Specifies a callback interface to call when the asynchronous operation has completed.

Syntax

C++

```
HRESULT SetCallback(  
    [in] IWSDAsyncCallback *pCallback,  
    [in] IUnknown          *pAsyncState  
);
```

Parameters

[in] pCallback

Pointer to a [IWSDAsyncCallback](#) object that contains the callback implemented by the user.

[in] pAsyncState

User-defined state information to pass to the callback.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pCallback</i> is NULL .

Remarks

The [IWSDAsyncCallback::AsyncOperationComplete](#) method is passed the result object associated with the completing message and the state.

pCallback is released when the [IWSDAsyncResult](#) object is destroyed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAsyncResult::SetWaitHandle method (wsdclient.h)

Article02/22/2024

Specifies a wait handle to set when the operation completes.

Syntax

```
C++

HRESULT SetWaitHandle(
    [in] HANDLE hWaitHandle
);
```

Parameters

[in] hWaitHandle

The wait handle to set.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>hWaitHandle</i> is NULL .
E_FAIL	The method failed.

Remarks

Do not close *hWaitHandle* until after the asynchronous operation has completed.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDAsyncResult](#)

IWSDAttachment interface (wsdattachment.h)

Article02/22/2024

Is the base interface for all other attachment types.

Inheritance

The IWSDAttachment interface inherits from the IUnknown interface.

Remarks

This interface is used strictly as a common ancestor for the various attachment types and has no members of its own.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)

See also

[IUnknown](#)

[IWSDInboundAttachment](#)

[IWSDOutboundAttachment](#)

IWSDDeviceHost interface (wsdhost.h)

Article10/05/2021

Represents a DPWS-compliant device . The device host will announce its presence on the network using the WS-Discovery protocol. The device host will also automatically respond to discovery queries and metadata requests.

The caller can register user-implemented services with the device host. These services will be exposed in the device metadata and the services will be available over the network. Messages bound for these services will be automatically dispatched into the service object.

Call [WSDCreateDeviceHost](#) or [WSDCreateDeviceHostAdvanced](#) to create an object that exposes this interface.

Inheritance

The `IWSDDeviceHost` interface inherits from the [IUnknown](#) interface. `IWSDDeviceHost` also has these types of members:

Methods

The `IWSDDeviceHost` interface has these methods.

 Expand table

IWSDDeviceHost::AddDynamicService
Registers a service object for incoming requests, but does not add the service to the device host metadata. This is used for transient (dynamic) services.
IWSDDeviceHost::Init
Initializes an instance of an <code>IWSDDeviceHost</code> object.
IWSDDeviceHost::RegisterPortType
Registers a port type for incoming messages.
IWSDDeviceHost::RegisterService
Registers a service object for incoming requests and adds the service to the device host metadata.

IWSDDeviceHost::RemoveDynamicService Unregisters a service object that was registered using AddDynamicService.
IWSDDeviceHost::RetireService Unregisters a service object that was registered using RegisterService and removes the service from the device host metadata.
IWSDDeviceHost::SetMetadata Sets the metadata for a device, excluding user-defined service metadata.
IWSDDeviceHost::SetServiceDiscoverable Controls whether or not the service is advertised using WS-Discovery.
IWSDDeviceHost::SignalEvent Notifies all subscribed clients that an event has occurred.
IWSDDeviceHost::Start Starts the device host and publishes the device host using a WS-Discovery Hello message.
IWSDDeviceHost::Stop Sends a WS-Discovery Bye message and stops the host.
IWSDDeviceHost::Terminate Terminates the host and releases any attached services.

Remarks

After retrieving this interface, the application would then:

1. Call the [RegisterPortType](#) method to register all necessary port types.
2. Call [SetMetadata](#) to describe the device and optionally call [RegisterService](#) one or more times to register services described in the service host metadata.
3. Call the [Start](#) method to start the device host and to publish the device using WS-Discovery. After starting the device host, you can optionally:
 - a. Call [AddDynamicService](#) for services not described in the service host metadata (for example, an ad hoc print job).

- b. Call [RetireService](#) to terminate action on and disconnect a service activated by the [RegisterService](#) method.
 - c. Call the [SignalEvent](#) method to indicate that notifications should be sent for subscriptions relating to a particular event.
4. Call the [Stop](#) method to terminate host execution and terminate publication of the device.

An **IWSDDeviceHost** object can provide an object for a service on demand (using a notification callback) when calling the host receives a request message directed at that service.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)

See also

[Overview of the WSDAPI Interfaces](#)

IWSDDeviceHost::AddDynamicService method (wsdhost.h)

Article 10/13/2021

Registers a service object for incoming requests, but does not add the service to the device host metadata. This is used for transient (dynamic) services.

Syntax

C++

```
HRESULT AddDynamicService(  
    [in] LPCWSTR pszServiceId,  
    [in, optional] LPCWSTR pszEndpointAddress,  
    [in, optional] const WSD_PORT_TYPE *pPortType,  
    [in, optional] const WSDXML_NAME *pPortName,  
    [in, optional] const WSDXML_ELEMENT *pAny,  
    [in, optional] IUnknown *pService  
);
```

Parameters

[in] pszServiceId

The ID for the dynamic service. The service ID must be distinct from all the service IDs in the service host metadata and from any other registered dynamic service. The *pszServiceId* must be a URI.

[in, optional] pszEndpointAddress

An optional URI to use as the endpoint address for this service. If none is specified, the device host will assume the service should be available on all local transport addresses.

[in, optional] pPortType

Reference to a [WSD_PORT_TYPE](#) structure that specifies the port type. May be **NULL**. Specify only one of *pPortType* and *pPortName*.

[in, optional] pPortName

Reference to a [WSDXML_NAME](#) structure that specifies the type of the service, with associating the service with a specified port. Specify only one of *pPortType* and *pPortName*.

[in, optional] pAny

Optional reference to an extensible section to be included in the dynamic service metadata.

[in, optional] pService

Optional reference to a host service object to register.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pszServiceId</i> is NULL .
E_INVALIDARG	The length in characters of <i>pszServiceId</i> or <i>pszEndpointAddress</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or both <i>pPortType</i> and <i>pPortName</i> are specified.
E_FAIL	The method failed. It may have failed because the host has not been initialized, or the service specified by <i>pszServiceId</i> could not be found. Call Init to initialize a device host.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

When this method is called, the device adds a reference to the service object and calls its methods in response to request messages addressed to the service. Call the [RemoveDynamicService](#) method on the device host to release its reference to the service and stop calling methods on the service.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::Init method (wsdhost.h)

Article 10/13/2021

Initializes an instance of an [IWSDDeviceHost](#) object, which is the host-side representation of a device.

Syntax

C++

```
HRESULT Init(  
    [in] LPCWSTR pszLocalId,  
    [in, optional] IWSDXMLContext *pContext,  
    [in, optional] IWSDAddress **ppHostAddresses,  
    [in, optional] DWORD dwHostAddressCount  
);
```

Parameters

[in] `pszLocalId`

The logical or physical address of the device. A logical address is of the form `urn:uuid:{guid}`. If *pszLocalId* is a logical address, the host will announce the logical address and then convert the address to a physical address when it receives Resolve or Probe messages.

If *pszLocalId* is a physical address (such as URL prefixed by http or https), the host will use the address as the physical address and will host on that address instead of the default one.

For secure communication, *pszLocalId* must be an URL prefixed by https, and the host will use the SSL/TLS protocol on the port specified in the URL. The recommended port is port 5358, as this port is reserved for secure connections with WSDAPI. If no port is specified, then the host will use port 443. The host port must be configured with an SSL server certificate. For more information about the configuration of host ports, see [HttpSetServiceConfiguration](#).

Any URL (http or https) must be terminated with a trailing slash. The URL must contain a valid IP address or hostname.

The following list shows some example values for *pszLocalId*. It is not a complete list of valid values.

- `http://192.168.0.1:5357/`
- `http://localhost/`
- `http://myHostname:5357/`

- `https://192.168.0.1:5358/`
- `https://myHostname/`
- `https://myHostname/myDevice/`
- `https://myHostname:5358/`

[in, optional] `pContext`

An [IWSDXMLContext](#) interface that defines custom message types or namespaces.

[in, optional] `ppHostAddresses`

A single [IWSDAddress](#) object or [IWSDTransportAddress](#) object. The objects provide information about specific addresses that the host should listen on.

If *pszLocalId* contains a local address, the resulting behavior is a mapping between the logical address and the supplied physical address (instead of a mapping between the logical address and the default physical address).

[in, optional] `dwHostAddressCount`

The number of items in the *ppHostAddresses* array. If *ppHostAddresses* is an [IWSDAddress](#) interface, count must be 1.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
<code>S_OK</code>	Method completed successfully.
<code>E_INVALIDARG</code>	<i>pszLocalId</i> is NULL , the length in characters of <i>pszLocalId</i> exceeds <code>WSD_MAX_TEXT_LENGTH</code> (8192), or the number of addresses referenced by <i>ppHostAddresses</i> does not match <i>dwHostAddressCount</i> .
<code>E_FAIL</code>	The device host is in an unexpected state.
<code>E_OUTOFMEMORY</code>	Insufficient memory to complete the operation.
<code>E_ABORT</code>	Initialization could not be completed.

Remarks

This method is called by [WSDCreateDeviceHost](#) and need not normally be called directly by your code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::RegisterPortType method (wsdhost.h)

Article10/13/2021

Registers a port type for incoming messages. All port types listed in the service host metadata must be registered.

Syntax

```
C++

HRESULT RegisterPortType(
    [in] const WSD_PORT_TYPE *pPortType
);
```

Parameters

[in] pPortType

Reference to a [WSD_PORT_TYPE](#) structure that describes the port type.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_OUTOFMEMORY	Insufficient memory to complete the operation.
S_FALSE	The port type specified by <i>pPortType</i> has already been registered.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::RegisterService method (wsdhost.h)

Article02/22/2024

Registers a service object for incoming requests and adds the service to the device host metadata.

Syntax

C++

```
HRESULT RegisterService(  
    [in] LPCWSTR pszServiceId,  
    [in] IUnknown *pService  
);
```

Parameters

[in] pszServiceId

The ID of the service to be registered. This ID must appear in the device's service host metadata.

[in] pService

The service object that will handle requests addressed to the specified service.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszServiceId</i> is NULL , the length in characters of <i>pszServiceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or a service matching <i>pszServiceId</i> has already been registered.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::RemoveDynamicService method (wsdhost.h)

Article02/22/2024

Unregisters a service object that was registered using [AddDynamicService](#). An unregistered service object does not receive incoming requests.

Syntax

C++

```
HRESULT RemoveDynamicService(  
    [in] LPCWSTR pszServiceId  
);
```

Parameters

[in] `pszServiceId`

The ID for the dynamic service to be removed.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszServiceId</i> is NULL , the length in characters of <i>pszServiceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or <i>pszServiceId</i> was not found in the list of dynamic services.
E_FAIL	The method failed. It may have failed because the host has not been initialized. Call Init to initialize a device host.

Remarks

The device host releases its reference to the service object after the service is unregistered. The service object will not receive callbacks after **RemoveDynamicService** has completed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::RetireService method (wsdhost.h)

Article02/22/2024

Unregisters a service object that was registered using [RegisterService](#) and removes the service from the device host metadata.

Syntax

C++

```
HRESULT RetireService(  
    [in] LPCWSTR pszServiceId  
);
```

Parameters

[in] `pszServiceId`

The ID of the service to be removed.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pszServiceId</i> is NULL .
E_INVALIDARG	The length in characters of <i>pszServiceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or <i>pszServiceId</i> was not found in the list of registered services.
E_FAIL	The method failed. It may have failed because the host has not been initialized. Call Init to initialize a device host.

Remarks

The device host releases its reference to the service object after the service is unregistered. The service object will not receive callbacks after **RetireService** has completed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::SetMetadata method (wsdhost.h)

Article 10/13/2021

Sets the metadata for a device, excluding user-defined service metadata.

Syntax

C++

```
HRESULT SetMetadata(  
    [in]          const WSD_THIS_MODEL_METADATA *pThisModelMetadata,  
    [in]          const WSD_THIS_DEVICE_METADATA *pThisDeviceMetadata,  
    [in, optional] const WSD_HOST_METADATA *pHostMetadata,  
    [in, optional] const WSD_METADATA_SECTION_LIST *pCustomMetadata  
);
```

Parameters

[in] pThisModelMetadata

Reference to a [WSD_THIS_MODEL_METADATA](#) structure which specifies metadata that is common to all instances of the model of this device. The **Manufacturer**, **ModelNames**, and **ModelNumber** members of the structure must contain non-NULL, non-blank entries.

[in] pThisDeviceMetadata

Reference to a [WSD_THIS_DEVICE_METADATA](#) structure which specifies metadata unique to this device. The **FriendlyName**, **FirmwareVersion**, and **SerialNumber** members of this structure must contain non-NULL, non-blank entries.

[in, optional] pHostMetadata

Reference to a [WSD_HOST_METADATA](#) structure that specifies service host metadata, which the specific data and characteristics of the device (for example, a printer supports color or has a stapler.).

[in, optional] pCustomMetadata

Reference to a [WSD_METADATA_SECTION_LIST](#) structure which specifies additional custom metadata associated with this device.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pThisDeviceMetadata</i> is NULL , <i>pThisModelMetadata</i> is NULL , or either structure does not contain the required members. See the parameter descriptions for a list of required members.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

This method must be called at least once prior to starting any device host which was registered with [RegisterService](#). It may be called after the device is started to update the metadata, in which case WS-Discovery Hello messages are issued indicating the new metadata version.

Note The update feature has not yet been implemented.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

IWSDDeviceHost::SetServiceDiscoverable method (wsdhost.h)

Article02/22/2024

Controls whether or not the service is advertised using WS-Discovery.

Syntax

C++

```
HRESULT SetServiceDiscoverable(  
    [in] LPCWSTR pszServiceId,  
    [in] BOOL     fDiscoverable  
);
```

Parameters

[in] pszServiceId

The ID for the service.

[in] fDiscoverable

TRUE if the service can be found using WS-Discovery, **FALSE** if the service is not visible to WS-Discovery.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszServiceId</i> is NULL , the length in characters of <i>pszServiceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or <i>pszServiceId</i> does not correspond to a registered service.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::SignalEvent method (wsdhost.h)

Article02/22/2024

Notifies all subscribed clients that an event has occurred.

Syntax

C++

```
HRESULT SignalEvent(  
    [in] LPCWSTR          pszServiceId,  
    [in] const void      *pBody,  
    [in] const WSD_OPERATION *pOperation  
);
```

Parameters

[in] pszServiceId

The ID of the service that generates the event.

[in] pBody

The body of the event.

[in] pOperation

Reference to a [WSD_OPERATION](#) structure that specifies the operation.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_FAIL	The host is not started. Call Start to start the device host.

E_INVALIDARG

pszServiceId is **NULL**, *pOperation* is **NULL**, the length in characters of *pszServiceId* exceeds WSD_MAX_TEXT_LENGTH (8192), there is no *ResponseType* structure associated with *pOperation*, or the service specified by *pszServiceId* is not subscribed to the event specified by the *ResponseType* member of *pOperation*.

Remarks

SignalEvent blocks until the event is sent to all clients. Since clients are contacted sequentially, it is possible that **SignalEvent** will block for a long time if any client responds slowly or is unreachable.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::Start method (wsdhost.h)

Article 02/22/2024

Starts the device host and publishes the device host using a WS-Discovery Hello message. If a notification sink is passed to this method, then the notification sink is also registered. After **Start** has been called successfully, the device host will automatically respond to Probe and Resolve messages.

Syntax

C++

```
HRESULT Start(  
    [in]          ULONGLONG          ullInstanceId,  
    [in]          const WSD_URI_LIST  *pScopeList,  
    [in, optional] IWSDDeviceHostNotify *pNotificationSink  
);
```

Parameters

[in] ullInstanceId

The instance identifier. If no identifier is provided, the current instance value + 1 is used as the default.

Note For compatibility with the WS-Discovery specification, this value must be less than or equal to UINT_MAX (4294967295).

[in] pScopeList

Scope of the device host. If **NULL**, no scopes are associated with the host.

[in, optional] pNotificationSink

Reference to an [IWSDDeviceHostNotify](#) object that specifies the notification sink.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
S_FALSE	The device host has already been started.
E_FAIL	The method failed. It may have failed because the host has not been initialized. Call Init to initialize a device host.
E_ABORT	There is no metadata associated with the host.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::Stop method (wsdhost.h)

Article02/22/2024

Sends a WS-Discovery [Bye](#) message and stops the host. After a host has been successfully stopped, it must be terminated with [IWSDDeviceHost::Terminate](#) before being released.

Syntax

```
C++  
  
HRESULT Stop();
```

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
S_FALSE	The host has already stopped.
E_FAIL	The host is not initialized or the host is not started.

Remarks

When a device host is stopped using this method, all services remain attached but no inbound messages are processed or otherwise handled.

Calling **Stop** is not necessary if the host has not been started.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHost::Terminate method (wsdhost.h)

Article02/22/2024

Terminates the host and releases any attached services. If a notification sink was passed to the [Start](#) method, then the notification sink is released.

Syntax

```
C++  
  
HRESULT Terminate();
```

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_FAIL	The host is uninitialized or the host has already been terminated.

Remarks

Services and notification sinks will not receive messages after the **Terminate** method has completed.

If this device host was started by calling [IWSDDeviceHost::Start](#), it must first be stopped by calling [IWSDDeviceHost::Stop](#) before **Terminate** can be called.

Terminate must be called before releasing the [IWSDDeviceHost](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHost](#)

IWSDDeviceHostNotify interface (wsdhost.h)

Article02/22/2024

Provides device-related notifications to an instance of an IWSDDeviceHost object.

Inheritance

The IWSDDeviceHostNotify interface inherits from the [IUnknown](#) interface. IWSDDeviceHostNotify also has these types of members:

Methods

The IWSDDeviceHostNotify interface has these methods.

 Expand table

IWSDDeviceHostNotify::GetService
Retrieves a service object that is not currently registered.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)

IWSDDeviceHostNotify::GetService method (wsdhost.h)

Article02/22/2024

Retrieves a service object that is not currently registered.

Syntax

C++

```
HRESULT GetService(  
    [in] LPCWSTR pszServiceId,  
    [out] IUnknown **ppService  
);
```

Parameters

[in] pszServiceId

The ID of the service to be produced.

[out] ppService

A reference to an [IWSDServiceProxy](#) object for the specified service.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceHostNotify](#)

IWSDDeviceProxy interface (wsdclient.h)

Article 10/05/2021

Represents a remote Devices Profile for Web Services (DPWS) device for client applications and middleware.

To get this interface, you can call [WSDCreateDeviceProxy](#).

Inheritance

The **IWSDDeviceProxy** interface inherits from the [IUnknown](#) interface. **IWSDDeviceProxy** also has these types of members:

Methods

The **IWSDDeviceProxy** interface has these methods.

 Expand table

IWSDDeviceProxy::BeginGetMetadata Sends an asynchronous request for metadata.
IWSDDeviceProxy::EndGetMetadata Ends an asynchronous request for metadata.
IWSDDeviceProxy::GetAllMetadata Retrieves all metadata for this device.
IWSDDeviceProxy::GetEndpointProxy Retrieves the endpoint proxy for the device.
IWSDDeviceProxy::GetHostMetadata Retrieves class-specific metadata for the device describing the features of the device and the services it hosts.
IWSDDeviceProxy::GetServiceProxyById Retrieves a generic IWSDServiceProxy service proxy by service ID.

IWSDDeviceProxy::GetServiceProxyByType
Retrieves a generic IWSDServiceProxy proxy for a service exposed by the device by port type name.
IWSDDeviceProxy::GetThisDeviceMetadata
Retrieves device-specific metadata for this device.
IWSDDeviceProxy::GetThisModelMetadata
Retrieves model-specific metadata for the device.
IWSDDeviceProxy::Init
Initializes the device proxy, optionally sharing a session with a previously initialized sponsoring device proxy.

Remarks

This interface is a client-side representation of a remote device. The proxy provides basic access to device metadata ([WSD_THIS_DEVICE_METADATA](#) and [WSD_THIS_MODEL_METADATA](#)), in addition to providing methods for creating service proxy objects. The service proxy objects correspond to service hosted on the device. For example, a television is a device and the tuner portion of the television is a service hosted on the device that has an accessible, atomic set of functions.

The **IWSDDeviceProxy** object exposes WSD-specific device semantics.

To use **IWSDDeviceProxy** in your client or middleware application:

1. Call [WSDCreateDeviceProxy](#).
2. Call any of the four metadata methods of the device proxy object.
3. Get an [IWSDServiceProxy](#) object, either by calling [GetServiceProxyById](#) or [GetServiceProxyByType](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

See also

[Overview of the WSDAPI Interfaces](#)

IWSDDeviceProxy::BeginGetMetadata method (wsdclient.h)

Article02/22/2024

Sends an asynchronous request for metadata.

Syntax

C++

```
HRESULT BeginGetMetadata(  
    [out] IWSDAsyncResult **ppResult  
);
```

Parameters

[out] ppResult

Returns an [IWSDAsyncResult](#) object that can be used to determine whether an operation has completed.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppResult</i> is NULL .
E_ABORT	The method could not be completed.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

BeginGetMetadata will force the device proxy to send a metadata request to the host. Once [EndGetMetadata](#) has been called, the results of the latest metadata retrieval are accessible through the [GetAllMetadata](#), [GetHostMetadata](#), [GetThisDeviceMetadata](#), and [GetThisModelMetadata](#) methods.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::EndGetMetadata method (wsdclient.h)

Article02/22/2024

Ends an asynchronous request for metadata and returns the metadata related to a device.

Syntax

C++

```
HRESULT EndGetMetadata(  
    [in] IWSDAsyncResult *pResult  
);
```

Parameters

[in] pResult

The [IWSDAsyncResult](#) object returned by [BeginGetMetadata](#).

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pResult</i> is NULL .
E_ABORT	The method could not be completed.
E_FAIL	The method failed. The metadata was not returned, was invalid, or a fault was generated.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

EndGetMetadata must only be called after the [IWSDAsyncResult](#) object returned by [BeginGetMetadata](#) has indicated that the operation is complete. Once EndGetMetadata has been called, the results of the latest metadata retrieval are accessible through the [GetAllMetadata](#), [GetHostMetadata](#), [GetThisDeviceMetadata](#), and [GetThisModelMetadata](#) methods.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetAllMetadata method (wsdclient.h)

Article10/13/2021

Retrieves all metadata for this device.

Syntax

```
C++

HRESULT GetAllMetadata(
    [out] WSD_METADATA_SECTION_LIST **ppMetadata
);
```

Parameters

[out] ppMetadata

Reference to a [WSD_METADATA_SECTION_LIST](#) structure that specifies all metadata related to a device. Do not release this object.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppMetadata</i> is NULL .

Remarks

This method is supplied so that extended metadata may be accessed. Manufacturer, service host and device-specific metadata is best obtained using methods provided specifically for those purposes.

GetAllMetadata will not cause the device proxy to retrieve metadata from the device. Instead, **GetAllMetadata** will return the metadata retrieved with the last call to [BeginGetMetadata](#) and [EndGetMetadata](#). If neither of these methods has been called, **GetAllMetadata** will return the metadata retrieved when the [IWSDDeviceProxy](#) object was initialized.

Upon success, the memory at *ppMetadata* will be valid until [BeginGetMetadata](#) or [EndGetMetadata](#) is called, or until the [IWSDDeviceProxy](#) object is released.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetEndpointProxy method (wsdclient.h)

Article02/22/2024

Retrieves the endpoint proxy for the device.

Syntax

```
C++

HRESULT GetEndpointProxy(
    [out] IWSEndpointProxy **ppProxy
);
```

Parameters

[out] ppProxy

An [IWSEndpointProxy](#) interface that implements a device services messaging proxy for this device.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppProxy</i> is NULL .

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetHostMetadata method (wsdclient.h)

Article02/22/2024

Retrieves class-specific metadata for the device describing the features of the device and the services it hosts.

Syntax

C++

```
HRESULT GetHostMetadata(  
    [out] WSD_HOST_METADATA **ppHostMetadata  
);
```

Parameters

[out] ppHostMetadata

Reference to a [WSD_HOST_METADATA](#) structure that specifies metadata. Do not release this object.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppHostMetadata</i> is NULL .

Remarks

GetHostMetadata will not cause the device proxy to retrieve metadata from the device. Instead, **GetHostMetadata** will return the metadata retrieved with the last call to

[BeginGetMetadata](#) and [EndGetMetadata](#). If neither of these methods has been called, **GetHostMetadata** will return the metadata retrieved when the [IWSDDeviceProxy](#) object was initialized.

Upon success, the memory at *ppMetadata* will be valid until [BeginGetMetadata](#) or [EndGetMetadata](#) is called or until the [IWSDDeviceProxy](#) object is released.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetServiceProxyById method (wsdclient.h)

Article02/22/2024

Retrieves a generic [IWSDServiceProxy](#) service proxy by service ID. Service IDs can be obtained by examining the service host metadata.

Syntax

```
C++

HRESULT GetServiceProxyById(
    [in] LPCWSTR          pszServiceId,
    [out] IWSDServiceProxy **ppServiceProxy
);
```

Parameters

[in] pszServiceId

The service ID.

[out] ppServiceProxy

Pointer to an [IWSDServiceProxy](#) object for the specified service proxy.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppServiceProxy</i> is NULL .
E_INVALIDARG	The length in characters of <i>pszServiceId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or there is no metadata associated with the service specified by <i>pszServiceId</i> .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

E_FAIL

There is no endpoint associated with the service proxy.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetServiceProxyByType method (wsdclient.h)

Article02/22/2024

Retrieves a generic [IWSDServiceProxy](#) proxy for a service exposed by the device by port type name.

Syntax

```
C++

HRESULT GetServiceProxyByType(
    [in] const WSDXML_NAME *pType,
    [out] IWSDServiceProxy **ppServiceProxy
);
```

Parameters

[in] pType

Reference to a [WSDXML_NAME](#) structure that specifies the port type name.

[out] ppServiceProxy

Pointer to the [IWSDServiceProxy](#) object associated with the specified service.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pType</i> or <i>ppServiceProxy</i> is NULL .
E_INVALIDARG	There is no metadata associated with the service specified by <i>pType</i> .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

E_FAIL

There is no endpoint associated with the service proxy.

Remarks

If the device hosts more than one service of the specified type, a proxy for any one of the services may be returned. In such a case, callers should not depend on any particular service proxy being returned.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetThisDeviceMetadata method (wsdclient.h)

Article02/22/2024

Retrieves device-specific metadata for this device.

Syntax

```
C++

HRESULT GetThisDeviceMetadata(
    [out] WSD_THIS_DEVICE_METADATA **ppThisDeviceMetadata
);
```

Parameters

[out] ppThisDeviceMetadata

Reference to a [WSD_THIS_DEVICE_METADATA](#) structure that specifies the device-specific metadata of this device. Do not release this object.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppThisDeviceMetadata</i> is NULL.

Remarks

GetThisDeviceMetadata will not cause the device proxy to retrieve metadata from the device. Instead, **GetThisDeviceMetadata** will return the metadata retrieved with the last call to [BeginGetMetadata](#) and [EndGetMetadata](#). If neither of these methods has been called,

GetThisDeviceMetadata will return the metadata retrieved when the [IWSDDeviceProxy](#) object was initialized.

Upon success, the memory at *ppMetadata* will be valid until [BeginGetMetadata](#) or [EndGetMetadata](#) is called or until the [IWSDDeviceProxy](#) object is released.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::GetThisModelMetadata method (wsdclient.h)

Article02/22/2024

Retrieves model-specific metadata for the device.

Syntax

```
C++

HRESULT GetThisModelMetadata(
    [out] WSD_THIS_MODEL_METADATA **ppManufacturerMetadata
);
```

Parameters

[out] ppManufacturerMetadata

Reference to a [WSD_THIS_MODEL_METADATA](#) structure that specifies manufacturer and model-specific metadata. Do not release this object.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppManufacturerMetadata</i> is NULL .

Remarks

GetThisModelMetadata will not cause the device proxy to retrieve metadata from the device. Instead, **GetThisModelMetadata** will return the metadata retrieved with the last call to [BeginGetMetadata](#) and [EndGetMetadata](#). If neither of these methods has been called,

GetThisModelMetadata will return the metadata retrieved when the [IWSDDeviceProxy](#) object was initialized.

Upon success, the memory at *ppMetadata* will be valid until [BeginGetMetadata](#) or [EndGetMetadata](#) is called or until the [IWSDDeviceProxy](#) object is released.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSDDeviceProxy::Init method (wsdclient.h)

Article 02/22/2024

Initializes the device proxy, optionally sharing a session with a previously initialized sponsoring device proxy.

Syntax

C++

```
HRESULT Init(  
    [in] LPCWSTR pszDeviceId,  
    [in] IWSDAddress *pDeviceAddress,  
    [in] LPCWSTR pszLocalId,  
    [in, optional] IWSDXMLContext *pContext,  
    [in, optional] IWSDDeviceProxy *pSponsor  
);
```

Parameters

[in] pszDeviceId

The logical address (ID) of the device.

[in] pDeviceAddress

Reference to an [IWSDAddress](#) object that contains the device configuration data.

[in] pszLocalId

The logical address of the client. The logical address is of the form, urn:uuid:{guid}. Used when the server needs to initiate a connection to the client.

[in, optional] pContext

Reference to an [IWSDXMLContext](#) object that defines custom message types or namespaces.

If **NULL**, a default context representing the built-in message types and namespaces is used.

[in, optional] pSponsor

Reference to an [IWSDDeviceProxy](#) object that is an optional device with which to share a session and lower layers.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszDeviceId</i> is NULL , <i>pszLocalId</i> is NULL , or the length in characters of either identifier string exceeds WSD_MAX_TEXT_LENGTH (8192).
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

This method is called by [WSDCreateDeviceProxy](#) and need not normally be called directly by the client code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDDeviceProxy](#)

IWSEndpointProxy interface (wsdclient.h)

Article 10/05/2021

Implements a device services messaging proxy.

Inheritance

The **IWSEndpointProxy** interface inherits from the [IUnknown](#) interface. **IWSEndpointProxy** also has these types of members:

Methods

The **IWSEndpointProxy** interface has these methods.

 Expand table

IWSEndpointProxy::AbortAsyncOperation
Aborts a pending asynchronous operation.
IWSEndpointProxy::GetErrorInfo
Retrieves information on the last error.
IWSEndpointProxy::GetFaultInfo
Retrieves information on the last received fault.
IWSEndpointProxy::ProcessFault
Processes a SOAP fault retrieved by GetFaultInfo.
IWSEndpointProxy::SendOneWayRequest
Sends a one-way request message.
IWSEndpointProxy::SendTwoWayRequest
Sends a two-way request message using a synchronous call pattern.
IWSEndpointProxy::SendTwoWayRequestAsync
Sends a two-way request message using an asynchronous call pattern.

Remarks

Service proxy objects may reside on multiple endpoints. An endpoint more completely represents a URL (contains additional useful data). One endpoint may support HTTP on IPv4 addresses and another may support HTTPS on IPv6 addresses. Since the same service lives on both endpoints, it is important that the service have underlying endpoint proxy objects, with each endpoint proxy corresponding to a single endpoint at which the service is available. The endpoint proxy takes care of simple messaging requests to the service, for example, sending one-way or two-way messages.

Endpoint proxies are generally used inside WSDAPI, but they can be retrieved from [IWSDServiceProxy](#) or [IWSDDeviceProxy](#) objects to expose message-level functionality.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

IWSEndpointProxy::AbortAsyncOperation method (wsdclient.h)

Article 10/13/2021

Aborts a pending asynchronous operation.

Syntax

C++

```
HRESULT AbortAsyncOperation(  
    [in] IWSEndpointProxy *pAsyncResult  
);
```

Parameters

[in] pAsyncResult

Calls the [Abort](#) method to end the asynchronous operation.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pAsyncResult</i> is NULL or <i>pAsyncResult</i> does not support the IWSEndpointProxy interface.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEndpointProxy](#)

IWSDEndpointProxy::GetErrorInfo method (wsdclient.h)

Article02/22/2024

Retrieves information on the last error.

Syntax

```
C++

HRESULT GetErrorInfo(
    [out] LPCWSTR *ppszErrorInfo
);
```

Parameters

[out] ppszErrorInfo

Pointer to a buffer containing the data for the last recorded error.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppszErrorInfo</i> is NULL .

Remarks

Note The error information returned in *ppszErrorInfo* must be released with **WSDFreeLinkedMemory** when it is no longer required for use.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEndpointProxy](#)

IWSDEndpointProxy::GetFaultInfo method (wsdclient.h)

Article02/22/2024

Retrieves information on the last received fault.

Syntax

```
C++

HRESULT GetFaultInfo(
    [out] WSD_SOAP_FAULT **ppFault
);
```

Parameters

[out] ppFault

Pointer to a pointer to a [WSD_SOAP_FAULT](#) structure containing the SOAP fault information.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppFault</i> is NULL.

Remarks

Note The fault information returned in *ppFault* must be released with **WSDFreeLinkedMemory** when it is no longer required for use.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEndpointProxy](#)

IWSDEndpointProxy::ProcessFault method (wsdclient.h)

Article02/22/2024

Processes a SOAP fault retrieved by [GetFaultInfo](#).

Syntax

C++

```
HRESULT ProcessFault(  
    [in] const WSD_SOAP_FAULT *pFault  
);
```

Parameters

[in] pFault

Pointer to a [WSD_SOAP_FAULT](#) structure containing the fault data.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pFault</i> is NULL.
S_FALSE	The fault was not stored in memory.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEndpointProxy](#)

IWSDEndpointProxy::SendOneWayRequest method (wsdclient.h)

Article02/22/2024

Sends a one-way request message.

Syntax

C++

```
HRESULT SendOneWayRequest(  
    [in] const void *pBody,  
    [in] const WSD_OPERATION *pOperation  
);
```

Parameters

[in] pBody

The body of the message.

[in] pOperation

Reference to a [WSD_OPERATION](#) structure that specifies the operation to perform.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pOperation</i> is NULL .

Remarks

This method is normally only called by generated proxy code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEndpointProxy](#)

IWSDEndpointProxy::SendTwoWayRequest method (wsdclient.h)

Article10/13/2021

Sends a two-way request message using a synchronous call pattern.

Syntax

```
C++

HRESULT SendTwoWayRequest(
    [in]          const void                *pBody,
    [in]          const WSD_OPERATION        *pOperation,
    [in, optional] const WSD_SYNCHRONOUS_RESPONSE_CONTEXT *pResponseContext
);
```

Parameters

[in] pBody

The body of the message.

[in] pOperation

Reference to a [WSD_OPERATION](#) structure that specifies the operation to perform.

[in, optional] pResponseContext

Reference to a [WSD_SYNCHRONOUS_RESPONSE_CONTEXT](#) structure or other context structure that specifies the context for handling the response to the request.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pOperation</i> is NULL .

Remarks

This method is normally only called by generated proxy code.

[WSD_SYNCHRONOUS_RESPONSE_CONTEXT](#) is used for the *responseContext* value when a synchronous call pattern is used.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSEndpointProxy](#)

IWSDEndpointProxy::SendTwoWayRequest Async method (wsdclient.h)

Article 02/22/2024

Sends a two-way request message using an asynchronous call pattern.

Syntax

C++

```
HRESULT SendTwoWayRequestAsync(  
    [in] const void *pBody,  
    [in] const WSD_OPERATION *pOperation,  
    [in] IUnknown *pAsyncState,  
    [in] IWSDAsyncCallback *pCallback,  
    [out] IWSDAsyncResult **pResult  
);
```

Parameters

[in] pBody

The body of the message.

[in] pOperation

Reference to a [WSD_OPERATION](#) structure that specifies the operation to perform.

[in] pAsyncState

Anonymous data passed to *pCallback* when the operation has completed. This data is used to associate a client object with the pending operation. This parameter may be optional.

[in] pCallback

Reference to an [IWSDAsyncCallback](#) object which performs the message status callback notification. This parameter may be optional.

[out] pResult

Reference to an [IWSDAsyncResult](#) object that specifies the results of the operation.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pOperation</i> or <i>pResult</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

This method is normally only called by generated proxy code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEndpointProxy](#)

IWSDEventingStatus interface (wsdclient.h)

Article02/22/2024

Implement this interface to receive notification when status changes occur in event subscriptions.

Inheritance

The IWSDEventingStatus interface inherits from the [IUnknown](#) interface. IWSDEventingStatus also has these types of members:

Methods

The IWSDEventingStatus interface has these methods.

 Expand table

IWSDEventingStatus::SubscriptionEnded
Called when the device terminated the subscription.
IWSDEventingStatus::SubscriptionRenewalFailed
Called when the subscription for the specified event action could not be renewed.
IWSDEventingStatus::SubscriptionRenewed
Called when the subscription for the specified event action was successfully renewed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

IWSDEventingStatus::SubscriptionEnded method (wsdclient.h)

Article02/22/2024

Called when the device terminated the subscription.

Syntax

C++

```
void SubscriptionEnded(  
    [in] LPCWSTR pszSubscriptionAction  
);
```

Parameters

[in] pszSubscriptionAction

URI of the event action.

Return value

None

Remarks

After an operation is subscribed to, the service proxy will listen for SubscriptionEnd messages from the subscription manager. If one is received for a specified subscription, **SubscriptionEnded** will be called to notify the client. **SubscriptionEnded** will not be called if the service proxy unsubscribes from the operation or if the host service goes offline.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEventingStatus](#)

IWSDEventingStatus::SubscriptionRenewed method (wsdclient.h)

Article02/22/2024

Called when the subscription for the specified event action was successfully renewed.

Syntax

C++

```
void SubscriptionRenewed(  
    [in] LPCWSTR pszSubscriptionAction  
);
```

Parameters

[in] pszSubscriptionAction

URI of the event action.

Return value

None

Remarks

After an operation is subscribed to, the service proxy will attempt to automatically renew the subscription until the client calls the appropriate **Unsubscribe** method or until the subscription is ended by the service. When the renewal succeeds, **SubscriptionRenewed** will be called to notify the client.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEventingStatus](#)

IWSDEventingStatus::SubscriptionRenewalFailed method (wsdclient.h)

Article 10/13/2021

Called when the subscription for the specified event action could not be renewed.

Syntax

C++

```
void SubscriptionRenewalFailed(  
    [in] LPCWSTR pszSubscriptionAction,  
    [in] HRESULT hr  
);
```

Parameters

[in] pszSubscriptionAction

URI of the event action.

[in] hr

HRESULT indicating the nature of the error.

Return value

None

Remarks

After an operation is subscribed to, the service proxy will attempt to automatically renew the subscription until the client calls the appropriate **Unsubscribe** method or until the subscription is ended by the service. When the renewal fails, **SubscriptionRenewalFailed** will be called to notify the client.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDEventingStatus](#)

IWSDHttpAddress interface (wsdbase.h)

Article02/22/2024

Provides access to the individual components of an HTTP address.

Inheritance

The **IWSDHttpAddress** interface inherits from [IWSDTransportAddress](#). **IWSDHttpAddress** also has these types of members:

Methods

The **IWSDHttpAddress** interface has these methods.

 Expand table

IWSDHttpAddress::GetPath
Gets the URI path for this address.
IWSDHttpAddress::GetSecure
Retrieves the status on whether TLS secure sessions are enabled for this address.
IWSDHttpAddress::SetPath
Sets the URI path for this address.
IWSDHttpAddress::SetSecure
Enables or disables TLS secure sessions for this address.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

See also

[IWSDTransportAddress](#)

IWSDHttpAddress::GetPath method (wsdbase.h)

Article02/22/2024

Gets the URI path for this address.

Syntax

```
C++

HRESULT GetPath(
    [out] LPCWSTR *ppszPath
);
```

Parameters

[out] ppszPath

Pointer to the URI path for this address. Do not release this object.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppszPath</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpAddress](#)

IWSDHttpAddress::GetSecure method (wsdbase.h)

Article02/22/2024

Retrieves the status on whether TLS secure sessions are enabled for this address.

Syntax

```
C++  
  
HRESULT GetSecure();
```

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	TLS secure sessions are enabled for this address.
S_FALSE	TLS secure sessions are disabled for this address.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpAddress](#)

IWSDHttpAddress::SetPath method (wsdbase.h)

Article02/22/2024

Sets the URI path for this address.

Syntax

```
C++

HRESULT SetPath(
    [in] LPCWSTR pszPath
);
```

Parameters

[in] pszPath

The URI path to use for this address.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The length in characters of <i>pszPath</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_FAIL	The method failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpAddress](#)

IWSDHttpAddress::SetSecure method (wsdbase.h)

Article02/22/2024

Enables or disables TLS secure sessions for this address.

Syntax

C++

```
HRESULT SetSecure(  
    [in] BOOL fSecure  
);
```

Parameters

[in] fSecure

TRUE to enable TLS secure session communications for this address, FALSE to disable TLS.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpAddress](#)

IWSDHttpAuthParameters interface (wsdbase.h)

Article02/22/2024

Use this interface to retrieve the access token or authorization scheme used during the authentication of a client.

Inheritance

The IWSDHttpAuthParameters interface inherits from the [IUnknown](#) interface. IWSDHttpAuthParameters also has these types of members:

Methods

The IWSDHttpAuthParameters interface has these methods.

 Expand table

IWSDHttpAuthParameters::GetAuthType
GetAuthType method retrieves the HTTP authentication scheme used during the authentication of the client.
IWSDHttpAuthParameters::GetClientAccessToken
GetClientAccessToken method retrieves the client access token that can be used to either authenticate or impersonate the client.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

See also

[Unknown](#)

IWSDHttpAuthParameters::GetClientAccessToken method (wsdbase.h)

Article02/22/2024

The **GetClientAccessToken** method retrieves the client access token that can be used to either authenticate or impersonate the client.

Syntax

C++

```
HRESULT GetClientAccessToken(  
    [out] HANDLE *phToken  
);
```

Parameters

[out] phToken

Pointer to a variable that on return receives the token handle.

Return value

This method returns S_OK on success.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

See also

IWSDHttpAuthParameters::GetAuthType method (wsdbase.h)

Article02/22/2024

The **GetAuthType** method retrieves the HTTP authentication scheme used during the authentication of the client.

Syntax

```
C++

HRESULT GetAuthType(
    PWSH_SECURITY_HTTP_AUTH_SCHEMES pAuthType
);
```

Parameters

pAuthType

Pointer to an [HTTP_REQUEST_AUTH_TYPE](#) value that indicates the HTTP authentication scheme used during authentication. Possible values include:

 Expand table

Value	Meaning
WSD_SECURITY_HTTP_AUTH_SCHEME_NEGOTIATE 0x1	Negotiate authentication.
WSD_SECURITY_HTTP_AUTH_SCHEME_NTLM 0x2	NTLM authentication.

Return value

This method returns S_OK on success.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

See also

[IWSDHttpAuthParameters](#)

IWSDHttpMessageParameters interface (wsdbase.h)

Article 10/05/2021

Provides access to the HTTP headers used when transmitting messages via SOAP-over-HTTP.

Inheritance

The `IWSDHttpMessageParameters` interface inherits from [IWSDMessageParameters](#).

`IWSDHttpMessageParameters` also has these types of members:

Methods

The `IWSDHttpMessageParameters` interface has these methods.

 Expand table

`IWSDHttpMessageParameters::Clear`

Clears the HTTP headers used for SOAP-over-HTTP transmissions.

`IWSDHttpMessageParameters::GetContext`

Retrieves the private transmission context for the current transaction.

`IWSDHttpMessageParameters::GetID`

Retrieves the transport ID for the current transaction.

`IWSDHttpMessageParameters::GetInboundHttpHeaders`

Retrieves the current HTTP headers used for inbound SOAP-over-HTTP transmissions.

`IWSDHttpMessageParameters::GetOutboundHttpHeaders`

Retrieves the current HTTP headers used for outbound SOAP-over-HTTP transmissions.

`IWSDHttpMessageParameters::SetContext`

Sets the private transmission context for the current transaction.

`IWSDHttpMessageParameters::SetID`

Sets the transport ID for the current transaction.
IWSDHttpMessageParameters::SetInboundHttpHeaders
Sets the HTTP headers used for inbound SOAP-over-HTTP transmissions.
IWSDHttpMessageParameters::SetOutboundHttpHeaders
Sets the HTTP headers used for outbound SOAP-over-HTTP transmissions.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

See also

[IWSDMessageParameters](#)

IWSDHttpMessageParameters::Clear method (wsdbase.h)

Article02/22/2024

Clears the HTTP headers used for SOAP-over-HTTP transmissions.

Syntax

```
C++  
  
HRESULT Clear();
```

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

IWSDHttpMessageParameters::GetContext method (wsdbase.h)

Article02/22/2024

Retrieves the private transmission context for the current transaction.

Syntax

```
C++

HRESULT GetContext(
    [out] IUnknown **ppContext
);
```

Parameters

[out] ppContext

Pointer to the pointer used to retrieve the desired private transmission context for the current transaction.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>ppContext</i> is NULL.
E_FAIL	Could not retrieve the context.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::GetID method (wsdbase.h)

Article02/22/2024

Retrieves the transport ID for the current transaction.

Syntax

```
C++

HRESULT GetID(
    [out] LPCWSTR *ppszId
);
```

Parameters

[out] ppszId

Pointer used to return the transport ID for the current transaction. Do not deallocate this pointer.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>ppszId</i> is NULL .
E_FAIL	The transport ID is not available.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::GetInboundHttpHeaders method (wsdbase.h)

Article02/22/2024

Retrieves the current HTTP headers used for inbound SOAP-over-HTTP transmissions.

Syntax

```
C++

HRESULT GetInboundHttpHeaders(
    [out] LPCWSTR *ppszHeaders
);
```

Parameters

[out] ppszHeaders

Pointer used to receive the current HTTP headers in use. Do not deallocate this pointer.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>ppszHeaders</i> is NULL .
E_FAIL	There are no headers available.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::GetOutboundHttpHeaders method (wsdbase.h)

Article02/22/2024

Retrieves the current HTTP headers used for outbound SOAP-over-HTTP transmissions.

Syntax

```
C++

HRESULT GetOutboundHttpHeaders(
    [out] LPCWSTR *ppszHeaders
);
```

Parameters

[out] ppszHeaders

Pointer used to receive the current HTTP headers in use. Do not deallocate this pointer.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>ppszHeaders</i> is NULL .
E_FAIL	There are no headers available.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::SetContext method (wsdbase.h)

Article02/22/2024

Sets the private transmission context for the current transaction.

Syntax

```
C++

HRESULT SetContext(
    [in] IUnknown *pContext
);
```

Parameters

[in] pContext

Pointer to the desired private transmission context for the current transaction.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::SetID method (wsdbase.h)

Article02/22/2024

Sets the transport ID for the current transaction.

Syntax

```
C++

HRESULT SetID(
    [in] LPCWSTR pszId
);
```

Parameters

[in] pszId

Pointer to the desired transport ID for the current transaction.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszId</i> is NULL or the length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::SetInboundHttpHeaders method (wsdbase.h)

Article02/22/2024

Sets the HTTP headers used for inbound SOAP-over-HTTP transmissions.

Syntax

C++

```
HRESULT SetInboundHttpHeaders(  
    [in] LPCWSTR pszHeaders  
);
```

Parameters

[in] pszHeaders

The HTTP headers to be set.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The length in characters of <i>pszHeaders</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDHttpMessageParameters::SetOutboundHttpHeaders method (wsdbase.h)

Article02/22/2024

Sets the HTTP headers used for outbound SOAP-over-HTTP transmissions.

Syntax

C++

```
HRESULT SetOutboundHttpHeaders(  
    [in] LPCWSTR pszHeaders  
);
```

Parameters

[in] pszHeaders

The HTTP headers to be set.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The length in characters of <i>pszHeaders</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDHttpMessageParameters](#)

IWSDInboundAttachment interface (wsdattachment.h)

Article07/27/2022

Allows applications to read MIME-encoded attachment data from an incoming message.

Inheritance

The IWSDInboundAttachment interface inherits from [IWSDAttachment](#).
IWSDInboundAttachment also has these types of members:

Methods

The IWSDInboundAttachment interface has these methods.

 Expand table

IWSDInboundAttachment::Close
Closes the current attachment MIME data stream. (IWSDInboundAttachment.Close)
IWSDInboundAttachment::Read
Retrieves attachment data from a message sent by a remote host.

Remarks

WSDAPI will provide an object implementing this interface when an attachment stream is received as part of a message.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)

See also

[IWSDAttachment](#)

IWSDInboundAttachment::Close method (wsdattachment.h)

Article02/22/2024

Closes the current attachment MIME data stream.

Syntax

```
C++  
  
HRESULT Close();
```

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

This method can be used to terminate the transfer of an incoming attachment while the transfer is in progress.

Usually, **Close** must be called before calling **Release()** on the [IWSDInboundAttachment](#) interface. The only time a **Close** call is not required is when [Read](#) returns S_FALSE, which indicates that the end of the attachment stream has been reached. In that case, simply call **Release()** on the **IWSDInboundAttachment** interface.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDInboundAttachment](#)

IWSDInboundAttachment::Read method (wsdattachment.h)

Article02/22/2024

Retrieves attachment data from a message sent by a remote host.

Syntax

C++

```
HRESULT Read(  
    [out] BYTE    *pBuffer,  
    [in]  DWORD   dwBytesToRead,  
    [out] LPDWORD pdwNumberOfBytesRead  
);
```

Parameters

[out] pBuffer

Pointer to a buffer receiving the data read from the attachment stream. The application program is responsible for allocating and freeing this data buffer.

[in] dwBytesToRead

Size of the *pBuffer* input buffer, in bytes.

[out] pdwNumberOfBytesRead

Pointer to a **DWORD** containing the number of bytes of data read from the attachment stream into the *pBuffer* input buffer.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

S_FALSE	The end of the attachment stream has been reached.
E_INVALIDARG	<i>pBuffer</i> is NULL .
E_POINTER	<i>pdwNumberOfBytesRead</i> is NULL .

Remarks

The **Read** method allows an application to receive arbitrary data from a remote host in a MIME-encapsulated message attachment. WSDAPI will provide an object implementing this interface when an attachment stream is received as part of a message. The call to **Read** opens the inbound attachment stream and transfers the attachment data to the application's buffer. If **Read** returns S_OK or S_FALSE, *pdwNumberOfBytesRead* is set to the number of bytes read, which may be less than the size of the buffer. The **Read** call may block on network traffic.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDInboundAttachment](#)

IWSDiscoveredService interface (wsddisco.h)

Article10/05/2021

This interface represents a remotely discovered host. WSDAPI returns this interface when calling [IWSDiscoveryProviderNotify::Add](#) and [IWSDiscoveryProviderNotify::Remove](#). The interface is populated when a match for an outstanding query issued using [IWSDiscoveryProvider::SearchByType](#), [IWSDiscoveryProvider::SearchByAddress](#), or [IWSDiscoveryProvider::SearchById](#) is received, or if a device on the network announces itself with a Hello message.

Inheritance

The **IWSDiscoveredService** interface inherits from the [IUnknown](#) interface. **IWSDiscoveredService** also has these types of members:

Methods

The **IWSDiscoveredService** interface has these methods.

 Expand table

IWSDiscoveredService::GetEndpointReference
Retrieves a WS-Addressing address referencing an endpoint of the remote device.
IWSDiscoveredService::GetExtendedDiscoXML
Retrieves custom or extensible data provided in the header or body of the SOAP message.
IWSDiscoveredService::GetInstanceId
Retrieves the instance identifier of this message.
IWSDiscoveredService::GetLocalInterfaceGUID
Retrieves the GUID of the local network interface over which the message was received.
IWSDiscoveredService::GetLocalTransportAddress
Retrieves the string representation of the local transport (IP) address.

IWSDiscoveredService::GetMetadataVersion
Retrieves the metadata version of this message.
IWSDiscoveredService::GetProbeResolveTag
Retrieves the search tag corresponding to this discovered service object.
IWSDiscoveredService::GetRemoteTransportAddress
Retrieves the string representation of the remote transport (IP) address.
IWSDiscoveredService::GetScopes
Retrieves a list of WS-Discovery Scopes.
IWSDiscoveredService::GetTypes
Retrieves a list of WS-Discovery Types.
IWSDiscoveredService::GetXAddrs
Retrieves a list of WS-Discovery XAddrs.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)

IWSDiscoveredService::GetEndpointReference method (wsddisco.h)

Article02/22/2024

Retrieves a WS-Addressing address referencing an endpoint of the remote device.

Syntax

```
C++

HRESULT GetEndpointReference(
    [out] WSD_ENDPOINT_REFERENCE **ppEndpointReference
);
```

Parameters

[out] ppEndpointReference

A WS-Addressing address referencing an endpoint of the remote device. For details, see [WSD_ENDPOINT_REFERENCE](#). Do not deallocate the output structure.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppEndPointReference</i> is NULL .

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetExtendedDiscoXML method (wsddisco.h)

Article02/22/2024

Retrieves custom or extensible data provided in the header or body of the SOAP message.

Syntax

C++

```
HRESULT GetExtendedDiscoXML(  
    [out] WSDXML_ELEMENT **ppHeaderAny,  
    [out] WSDXML_ELEMENT **ppBodyAny  
);
```

Parameters

[out] ppHeaderAny

Custom data added to the header portion of the SOAP message. For details, see [WSDXML_ELEMENT](#). Do not deallocate the output structure.

[out] ppBodyAny

Custom data added to the body portion of the SOAP message. For details, see [WSDXML_ELEMENT](#). Do not deallocate the output structure.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

Some devices may add custom data to the header and body portions of the SOAP message to convey additional information in the discovery phase.

The resulting pointer values are only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetInstanceId method (wsddisco.h)

Article02/22/2024

Retrieves the instance identifier of this message.

Syntax

C++

```
HRESULT GetInstanceId(  
    [out] ULONGLONG *pullInstanceId  
);
```

Parameters

[out] pullInstanceId

A pointer to the instance identifier of this message.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pullInstanceId</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetLocalInterfaceGUID method (wsddisco.h)

Article02/22/2024

Retrieves the GUID of the local network interface over which the message was received.

Syntax

```
C++

HRESULT GetLocalInterfaceGUID(
    [out] GUID *pGuid
);
```

Parameters

[out] pGuid

GUID of the local network interface over which the message was received. Structure will be cleared if the local interface GUID is not available.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pGuid</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetLocalTransportAddress method (wsddisco.h)

Article 10/13/2021

Retrieves the string representation of the local transport (IP) address.

Syntax

C++

```
HRESULT GetLocalTransportAddress(  
    [out] LPCWSTR *ppszLocalTransportAddress  
);
```

Parameters

[out] ppszLocalTransportAddress

String representation of the local transport (IP) address. Is **NULL** if not available. Do not deallocate the output string.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppszLocalTransportAddress</i> is NULL .

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetMetadataVersion method (wsddisco.h)

Article02/22/2024

Retrieves the metadata version of this message.

Syntax

```
C++

HRESULT GetMetadataVersion(
    [out] ULONGLONG *pullMetadataVersion
);
```

Parameters

[out] pullMetadataVersion

Metadata version of this message.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pullMetadataVersion</i> is NULL .

Remarks

Versioning is used to determine if metadata exchange should be performed again due to a state change on the device.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetProbeResolveTag method (wsddisco.h)

Article02/22/2024

Retrieves the search tag corresponding to this discovered service object.

Syntax

C++

```
HRESULT GetProbeResolveTag(  
    [out] LPCWSTR *ppszTag  
);
```

Parameters

[out] ppszTag

Search tag passed to the [IWSDiscoveryProvider](#) search method that this [IWSDiscoveredService](#) object corresponds to. If the [IWSDiscoveredService](#) is the result of a Hello message, the tag is **NULL**. Do not deallocate the output string.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppszTag</i> is NULL .

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetRemoteTransportAddress method (wsddisco.h)

Article10/13/2021

Retrieves the string representation of the remote transport (IP) address.

Syntax

C++

```
HRESULT GetRemoteTransportAddress(  
    [out] LPCWSTR *ppszRemoteTransportAddress  
);
```

Parameters

[out] ppszRemoteTransportAddress

String representation of the remote transport (IP) address. Is **NULL** if not available. Do not deallocate the output string.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppszRemoteTransportAddress</i> is NULL .

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

The string returned by this method may contain an IPv4 or unbracketed IPv6 address such as "fe80::1". It may also contain a bracketed IPv6 address that includes the port such as "

[fe80::1]:1234". The caller should parse the string carefully to account for both possibilities.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetScopes method (wsddisco.h)

Article02/22/2024

Retrieves a list of WS-Discovery Scopes.

Syntax

```
C++

HRESULT GetScopes(
    [out] WSD_URI_LIST **ppScopesList
);
```

Parameters

[out] ppScopesList

List of WS-Discovery Scopes provided in the Hello, ProbeMatch, or ResolveMatch message sent by the remote device. For details, see [WSD_URI_LIST](#). Do not deallocate the output structure.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppScopesList</i> is NULL .

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetTypes method (wsddisco.h)

Article10/13/2021

Retrieves a list of WS-Discovery Types.

Syntax

```
C++

HRESULT GetTypes(
    [in] WSD_NAME_LIST **ppTypesList
);
```

Parameters

[in] ppTypesList

List of WS-Discovery Types provided in the Hello, ProbeMatch, or ResolveMatch message sent by the remote device. For details, see [WSD_NAME_LIST](#). Do not deallocate the output structure.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppTypesList</i> is NULL .

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveredService::GetXAddrs method (wsddisco.h)

Article10/13/2021

Retrieves a list of WS-Discovery XAddrs.

Syntax

C++

```
HRESULT GetXAddrs(  
    [out] WSD_URI_LIST **ppXAddrsList  
);
```

Parameters

[out] ppXAddrsList

List of WS-Discovery XAddrs provided in the Hello, ProbeMatch, or ResolveMatch message sent by the remote device. For details, see [WSD_URI_LIST](#). Do not deallocate the output structure.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	ppXAddrsList is NULL.

Remarks

The resulting pointer value is only valid for the lifetime of the [IWSDiscoveredService](#) object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveredService](#)

IWSDiscoveryProvider interface (wsddisco.h)

Article10/05/2021

This interface is used to discover services on the network advertised by WS-Discovery.

To get this interface, you can call [WSDCreateDiscoveryProvider](#).

Inheritance

The **IWSDiscoveryProvider** interface inherits from the [IUnknown](#) interface. **IWSDiscoveryProvider** also has these types of members:

Methods

The **IWSDiscoveryProvider** interface has these methods.

 Expand table

IWSDiscoveryProvider::Attach
Attaches a callback interface to the discovery provider.
IWSDiscoveryProvider::Detach
Detaches a callback interface from the discovery provider.
IWSDiscoveryProvider::GetXMLContext
Gets the XML context associated with this provider.
IWSDiscoveryProvider::SearchByAddress
Initializes a search for WS-Discovery hosts by device address.
IWSDiscoveryProvider::SearchById
Initializes a search for WS-Discovery hosts by device identifier.
IWSDiscoveryProvider::SearchByType
Initializes a search for WS-Discovery hosts by device type.

[IWSDiscoveryProvider::SetAddressFamily](#)

Specifies the IP address family (IPv4, IPv6, or both) to search when discovering WSD devices.

Remarks

The Discovery Provider represents the "client" view of WS-Discovery.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)

See also

[Overview of the WSDAPI Interfaces](#)

IWSDiscoveryProvider::Attach method (wsddisco.h)

Article02/22/2024

Attaches a callback interface to the discovery provider.

Syntax

C++

```
HRESULT Attach(  
    [in] IWSDiscoveryProviderNotify *pSink  
);
```

Parameters

[in] pSink

Interface to receive callback notifications. Search results as well as the Hello and Bye messages are communicated to this interface via the callbacks.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pSink</i> is NULL .
E_ABORT	A callback interface has already been attached to the provider.

Remarks

Note Attach must be called before any other IWSDiscoveryProvider method is used, except for SetAddressFamily.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProvider](#)

IWSDiscoveryProvider::Detach method (wsddisco.h)

Article02/22/2024

Detaches a callback interface from the discovery provider.

Syntax

```
C++

HRESULT Detach();
```

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_ABORT	A callback interface has not been attached. You must call Attach before calling this method.

Remarks

If a callback interface has been attached to a discovery provider via the [Attach](#) method, then **Detach** must be called before releasing the reference to the provider interface object.

The **Detach** operation blocks until all callbacks into the associated [IWSDiscoveryProviderNotify](#) object have completed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProvider](#)

IWSDiscoveryProvider::GetXMLContext method (wsddisco.h)

Article02/22/2024

Gets the XML context associated with this provider.

Syntax

```
C++

HRESULT GetXMLContext(
    [out] IWSDXMLContext **ppContext
);
```

Parameters

[out] ppContext

Pointer to a pointer variable containing the XML context.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>ppContext</i> is NULL .
E_ABORT	The discovery provider has not been created. Call WSDCreateDiscoveryProvider to create a provider.

Remarks

Returns an optional context for the XML state of the transaction. If the service layer is used then this should be the context the XML namespaces and types were registered with.

Note [Attach](#) must be called before any other [IWSDiscoveryProvider](#) method is used.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDXMLContext](#)

[IWSDiscoveryProvider](#)

IWSDiscoveryProvider::SearchByAddress method (wsddisco.h)

Article10/13/2021

Initializes a search for [WS-Discovery](#) hosts by device address.

Syntax

```
C++

HRESULT SearchByAddress(
    [in]          LPCWSTR pszAddress,
    [in, optional] LPCWSTR pszTag
);
```

Parameters

[in] pszAddress

The HTTP transport address of the device.

[in, optional] pszTag

Optional identifier tag for this search. May be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszAddress</i> is NULL , the length in characters of <i>pszAddress</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or the length in characters of <i>pszTag</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_ABORT	A callback interface has not been attached. You must call Attach before calling this method.
E_OUTOFMEMORY	Not enough memory exists to perform the operation.

Remarks

SearchByAddress initiates a WS-Discovery [Probe](#) over HTTP in an attempt to identify a device at a known URL. The Probe is sent to the address specified by *pszAddress*. This call may result in one or more [Add](#) callbacks. If any **Add** callbacks are issued before the search completes, a [SearchComplete](#) callback will be issued; otherwise, a [SearchFailed](#) callback will be issued. The interval between initiating the search and receiving either of these notifications can be up to 30 seconds.

pszTag is an optional user provided string which will be fed back in either callback, allowing the caller to associate the callback with the original query.

For information about troubleshooting applications calling this method, see [Troubleshooting WSDAPI Applications](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProvider](#)

IWSDiscoveryProvider::SearchById method (wsddisco.h)

Article02/22/2024

Initializes a search for [WS-Discovery](#) hosts by device identifier.

Syntax

```
C++

HRESULT SearchById(
    [in] LPCWSTR pszId,
    [in, optional] LPCWSTR pszTag
);
```

Parameters

[in] pszId

Device identifier of the desired discovery provider.

[in, optional] pszTag

Optional identifier tag for this search. May be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pszId</i> is NULL , the length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192), or the length in characters of <i>pszTag</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_ABORT	A callback interface has not been attached. You must call Attach before calling this method.
E_OUTOFMEMORY	Not enough memory exists to perform the operation.

Remarks

SearchById initiates a WS-Discovery [Resolve](#) in an attempt to locate a previously known specific device. *pszId* is used as the endpoint address in the Resolve. This call may result in one or more [Add](#) callbacks. If any **Add** callbacks are issued before the search completes, a [SearchComplete](#) callback will be issued; otherwise, a [SearchFailed](#) callback will be issued.

pszTag is an optional user provided string which will be fed back in either callback, allowing the caller to associate the callback with the original query.

For information about troubleshooting applications calling this method, see [Troubleshooting WSDAPI Applications](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProvider](#)

IWSDiscoveryProvider::SearchByType method (wsddisco.h)

Article 02/22/2024

Initializes a search for [WS-Discovery](#) hosts by device type.

Syntax

C++

```
HRESULT SearchByType(  
    [in, optional] const WSD_NAME_LIST *pTypesList,  
    [in, optional] const WSD_URI_LIST *pScopesList,  
    [in, optional] LPCWSTR pszMatchBy,  
    [in, optional] LPCWSTR pszTag  
);
```

Parameters

[in, optional] pTypesList

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of discovery provider types to search for. May be **NULL**.

[in, optional] pScopesList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of discovery provider scopes to search for. May be **NULL**.

[in, optional] pszMatchBy

Matching rule used for scopes. May be **NULL**.

[in, optional] pszTag

Optional identifier tag for this search. May be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The length in characters of <i>pszMatchBy</i> exceeds WSD_MAX_TEXT_LENGTH (8192) or the length in characters of <i>pszTag</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_ABORT	A callback interface has not been attached. You must call Attach before calling this method.
E_OUTOFMEMORY	Not enough memory exists to perform the operation.

Remarks

SearchByType initiates a WS-Discovery [Probe](#) in an attempt to locate discovery hosts matching the provided criteria. This method allows matching by types, scopes, some combination of the two, or matching all discovery capable devices (when no scopes or types are provided).

pszMatchBy should be provided if and only if *pScopesList* is also provided. This call may result in one or more [Add](#) callbacks. If any **Add** callbacks are issued before the search completes, a [SearchComplete](#) callback will be issued; otherwise, a [SearchFailed](#) callback will be issued.

pszTag is an optional user provided string which will be fed back in either callback, allowing the caller to associate the callback with the original query.

For information about troubleshooting applications calling this method, see [Troubleshooting WSDAPI Applications](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProvider](#)

IWSDiscoveryProvider::SetAddressFamily method (wsddisco.h)

Article02/22/2024

Specifies the IP address family (IPv4, IPv6, or both) to search when discovering WSD devices.

Syntax

C++

```
HRESULT SetAddressFamily(  
    [in] DWORD dwAddressFamily  
);
```

Parameters

[in] dwAddressFamily

The address family to search when discovering devices.

 Expand table

Value	Meaning
WSDAPI_ADDRESSFAMILY_IPV4	Search over IPv4 addresses.
WSDAPI_ADDRESSFAMILY_IPV6	Search over IPv6 addresses.
WSDAPI_ADDRESSFAMILY_IPV4 WSDAPI_ADDRESSFAMILY_IPV6	Search over IPv4 and IPv6 addresses.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

E_INVALIDARG	<i>dwAddressFamily</i> has a value other than WSDAPI_ADDRESSFAMILY_IPV4, WSDAPI_ADDRESSFAMILY_IPV6, or WSDAPI_ADDRESSFAMILY_IPV4 WSDAPI_ADDRESSFAMILY_IPV6.
STG_E_INVALIDFUNCTION	The address family has already been set for this publisher.
HRESULT_FROM_WIN32(WSAEsocktnosupport)	The system does not support the address family specified by <i>dwAddressFamily</i> .

Remarks

This method can be called only once on a provider. This method must be called before a notification sink is attached to the provider. That means **SetAddressFamily** must be called before [Attach](#) is called on a provider.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h
DLL	Wsdapi.dll

See also

[IWSDiscoveryProvider](#)

IWSDiscoveryProviderNotify interface (wsddisco.h)

Article10/05/2021

Is implemented by the client program to receive callback notifications from [IWSDiscoveryProvider](#).

Inheritance

The `IWSDiscoveryProviderNotify` interface inherits from the [IUnknown](#) interface. `IWSDiscoveryProviderNotify` also has these types of members:

Methods

The `IWSDiscoveryProviderNotify` interface has these methods.

 Expand table

IWSDiscoveryProviderNotify::Add Provides information on either a newly announced discovery host (from a Hello message), or a match to a user initiated query.
IWSDiscoveryProviderNotify::Remove Provides information on a recently departed discovery host (from a Bye message).
IWSDiscoveryProviderNotify::SearchComplete Called to indicate a user initiated search has successfully completed and no more matches for the search will be accepted.
IWSDiscoveryProviderNotify::SearchFailed Is called to indicate a user initiated search has failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)

IWSDiscoveryProviderNotify::Add method (wsddisco.h)

Article 02/22/2024

Provides information on either a newly announced discovery host (from a Hello message), or a match to a user initiated query.

Syntax

C++

```
HRESULT Add(  
    [in] IWSDiscoveredService *pService  
);
```

Parameters

[in] pService

A pointer to an [IWSDiscoveredService](#) interface that represents a remote discovery host.

Return value

The return value is not meaningful. An implementer should return S_OK.

Remarks

Add will be called once for each successful match to an outstanding query, and once per announcement of a new discovery host.

Note Multiple simultaneous calls may be made to **Add** by the provider, so it is essential that shared data be synchronized when implementing this callback routine.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProviderNotify](#)

IWSDiscoveryProviderNotify::Remove method (wsddisco.h)

Article 02/22/2024

Provides information on a recently departed discovery host (from a Bye message).

Syntax

C++

```
HRESULT Remove(  
    [in] IWSDiscoveredService *pService  
);
```

Parameters

[in] pService

A pointer to an [IWSDiscoveredService](#) interface that represents a remote discovery host.

Return value

The return value is not meaningful. An implementer should return S_OK.

Remarks

Remove will be called once per announcement of a departing discovery host.

Note Multiple simultaneous calls may be made to **Remove** by the provider, so it is essential that shared data be synchronized when implementing this callback routine.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProviderNotify](#)

IWSDiscoveryProviderNotify::SearchComplete method (wsddisco.h)

Article 02/22/2024

Called to indicate a user initiated search has successfully completed and no more matches for the search will be accepted.

Syntax

C++

```
HRESULT SearchComplete(  
    [in, optional] LPCWSTR pszTag  
);
```

Parameters

[in, optional] pszTag

Search tag passed to the [IWSDiscoveryProvider](#) search method.

Return value

The return value is not meaningful. An implementer should return S_OK.

Remarks

If no responses are received for a given search, then [IWSDiscoveryProviderNotify::SearchFailed](#) will be called to indicate this.

The interval between initiating the search with [SearchByType](#) or [SearchById](#) and receiving a **SearchComplete** notification is a maximum of 10 seconds, based on MATCH_TIMEOUT from [WS-Discovery](#) and amended by the [DPWS Appendix I](#). The interval between initiating the search with [SearchByAddress](#) and receiving a **SearchComplete** notification is a maximum of 150 seconds.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProviderNotify](#)

IWSDiscoveryProviderNotify::SearchFailed method (wsddisco.h)

Article 02/22/2024

Is called to indicate a user initiated search has failed.

Syntax

C++

```
HRESULT SearchFailed(  
    [in]          HRESULT hr,  
    [in, optional] LPCWSTR pszTag  
);
```

Parameters

[in] hr

Cause of the search failure which initiated this callback. A value of **S_FALSE** indicates the search completed without issuing any Add callbacks.

[in, optional] pszTag

Optional identifier tag for this search. May be **NULL**.

Return value

The return value is not meaningful. An implementer should return **S_OK**.

Remarks

[SearchComplete](#) is called if any responses were successfully received.

SearchFailed is called if a user initiated query does not result in a response. In this case, the value of the *hr* parameter will be **S_FALSE**. **SearchFailed** can optionally be called if errors occur in the attempted transmission of the query, since query transmission is not necessarily synchronous. *pszTag* will match the user supplied tag from the query, and should be used to identify which query failed.

The interval between initiating the search with [SearchByType](#) or [SearchById](#) and receiving a **SearchFailed** notification is a maximum of 10 seconds, based on MATCH_TIMEOUT from [WS-Discovery](#) and amended by the [DPWS Appendix I](#). The interval between initiating the search with [SearchByAddress](#) and receipt of a **SearchFailed** notification is typically 21 seconds, but can be a maximum of 150 seconds.

Note Multiple simultaneous calls may be made to **SearchFailed** by the provider, so it is essential that shared data be synchronized in this callback.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryProviderNotify](#)

IWSDiscoveryPublisher interface (wsddisco.h)

Article 10/05/2021

Provides methods for announcing hosts and managing incoming queries to hosts.

To get this interface, call [WSDCreateDiscoveryPublisher](#).

Inheritance

The **IWSDiscoveryPublisher** interface inherits from the [IUnknown](#) interface.

IWSDiscoveryPublisher also has these types of members:

Methods

The **IWSDiscoveryPublisher** interface has these methods.

 Expand table

[IWSDiscoveryPublisher::GetXMLContext](#)

Gets the XML context associated with the device.

[IWSDiscoveryPublisher::MatchProbe](#)

Determines whether a Probe message matches the specified host and sends a WS-Discovery ProbeMatches message if the match is made.

[IWSDiscoveryPublisher::MatchProbeEx](#)

Determines whether a Probe message matches the specified host and sends a WS-Discovery ProbeMatches message with extended information if the match is made.

[IWSDiscoveryPublisher::MatchResolve](#)

Determines whether a Resolve message matches the specified host and sends a WS-Discovery ResolveMatches message if the match is made.

[IWSDiscoveryPublisher::MatchResolveEx](#)

Determines whether a Resolve message matches the specified host and sends a WS-Discovery ResolveMatches message with extended information if the match is made.

IWSDiscoveryPublisher::Publish
Announces the presence of a network host by sending a Hello message.
IWSDiscoveryPublisher::PublishEx
Announces the presence of a network host by sending a Hello message with extended information.
IWSDiscoveryPublisher::RegisterNotificationSink
Attaches a callback notification sink to the discovery publisher.
IWSDiscoveryPublisher::RegisterScopeMatchingRule
Adds support for a custom scope matching rule.
IWSDiscoveryPublisher::SetAddressFamily
Specifies the IP address family (IPv4, IPv6, or both) over which the host will be published.
IWSDiscoveryPublisher::UnPublish
Announces the departure of a network host by sending a Bye message.
IWSDiscoveryPublisher::UnRegisterNotificationSink
Detaches a callback notification sink from the discovery publisher.
IWSDiscoveryPublisher::UnRegisterScopeMatchingRule
Removes support for a custom scope matching rule.

Remarks

This interface represents the "server" or "host" side of [WS-Discovery](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)

See also

[Overview of the WSDAPI Interfaces](#)

IWSDiscoveryPublisher::GetXMLContext method (wsddisco.h)

Article02/22/2024

Gets the XML context associated with the device.

Syntax

```
C++

HRESULT GetXMLContext(
    [out] IWSDXMLContext **ppContext
);
```

Parameters

[out] ppContext

Pointer to a pointer to an [IWSDXMLContext](#) object that represents the XML context.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	<i>ppContext</i> is NULL .
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .

Remarks

Returns an optional context for the XML state of the transaction. If the service layer is used then this should be the context the XML namespaces and types were registered with.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDXMLContext](#)

[IWSDiscoveryPublisher](#)

IWSDDiscoveryPublisher::MatchProbe method (wsddisco.h)

Article 10/13/2021

Determines whether a [Probe](#) message matches the specified host and sends a WS-Discovery [ProbeMatches](#) message if the match is made.

Syntax

C++

```
HRESULT MatchProbe(  
    [in]          const WSD_SOAP_MESSAGE *pProbeMessage,  
    [in]          IWSDMessageParameters *pMessageParameters,  
    [in]          LPCWSTR                pszId,  
    [in]          ULONGLONG             ullMetadataVersion,  
    [in]          ULONGLONG             ullInstanceId,  
    [in]          ULONGLONG             ullMessageNumber,  
    [in, optional] LPCWSTR                pszSessionId,  
    [in, optional] const WSD_NAME_LIST    *pTypesList,  
    [in, optional] const WSD_URI_LIST     *pScopesList,  
    [in, optional] const WSD_URI_LIST     *pXAddrsList  
);
```

Parameters

[in] pProbeMessage

Pointer to a [WSD_SOAP_MESSAGE](#) structure that represents the Probe message passed to the notification sink's [ProbeHandler](#).

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) object that represents the transmission parameters passed in to the notification sink's [ProbeHandler](#).

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by http or https, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] ullMetadataVersion

The current metadata version.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in] ullInstanceId

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in] ullMessageNumber

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in, optional] pszSessionId

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the AppSequence block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#).

This parameter may be **NULL**.

[in, optional] pTypesList

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of types supported by the publishing host. May be **NULL**.

If *pTypesList* is specified, **MatchProbe** will use WS-Discovery matching logic to verify that the types in the list match the types specified in *pProbeMessage*.

[in, optional] pScopesList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of matching scopes supported by the publishing host. The list contains hash values in string form. May be **NULL**.

If *pScopesList* is specified, **MatchProbe** will use WS-Discovery matching logic to verify that the scopes in the list match the scopes specified in *pProbeMessage*.

[in, optional] pXAddrsList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of transport addresses supported by the publishing host. Each transport address string contains an address and port number which can be used for connection by a remote host. May be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: • <i>pszId</i> is NULL. • The length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). • The length in characters of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). • <i>pProbeMessage</i> is NULL.
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

MatchProbe should be called only when the discovery publisher has issued a [ProbeHandler](#) callback. *pProbeMessage* and *pMessageParameters* are passed directly from the callback into **MatchProbe**. The **ProbeHandler** also passes information required by the publisher to

determine if the supplied Probe message matches and, if so, to issue a ProbeMatches response if appropriate.

MatchProbe sends ProbeMatches messages on all bound adapters and automatically issues message retransmissions when required by WS-Discovery.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDDiscoveryPublisher::MatchProbeEx method (wsddisco.h)

Article 10/13/2021

Determines whether a [Probe](#) message matches the specified host and sends a WS-Discovery [ProbeMatches](#) message with extended information if the match is made.

Syntax

C++

```
HRESULT MatchProbeEx(  
    [in]          const WSD_SOAP_MESSAGE *pProbeMessage,  
    [in]          IWSDMessageParameters *pMessageParameters,  
    [in]          LPCWSTR                pszId,  
    [in]          ULONGLONG             ullMetadataVersion,  
    [in]          ULONGLONG             ullInstanceId,  
    [in]          ULONGLONG             ullMessageNumber,  
    [in, optional] LPCWSTR                pszSessionId,  
    [in, optional] const WSD_NAME_LIST    *pTypesList,  
    [in, optional] const WSD_URI_LIST     *pScopesList,  
    [in, optional] const WSD_URI_LIST     *pXAddrsList,  
    [in, optional] const WSDXML_ELEMENT   *pHeaderAny,  
    [in, optional] const WSDXML_ELEMENT   *pReferenceParameterAny,  
    [in, optional] const WSDXML_ELEMENT   *pPolicyAny,  
    [in, optional] const WSDXML_ELEMENT   *pEndpointReferenceAny,  
    [in, optional] const WSDXML_ELEMENT   *pAny  
);
```

Parameters

[in] pProbeMessage

Pointer to a [WSD_SOAP_MESSAGE](#) structure that represents the Probe message passed to the notification sink's ProbeHandler.

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) object that represents the transmission parameters passed in to the notification sink's ProbeHandler.

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by `http` or `https`, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] `ullMetadataVersion`

Current metadata version.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in] `ullInstanceId`

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in] `ullMessageNumber`

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in, optional] `pszSessionId`

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the `AppSequence` block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#).

This parameter may be **NULL**.

[in, optional] `pTypesList`

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of types supported by the publishing host. May be **NULL**.

If *pTypesList* is specified, [MatchProbe](#) will use WS-Discovery matching logic to verify that the types in the list match the types specified in *pProbeMessage*.

[in, optional] *pScopesList*

Pointer to a [WSD_URI_LIST](#) structure that represents the list of matching scopes supported by the publishing host. The list contains hash values in string form. May be **NULL**.

If *pScopesList* is specified, [MatchProbe](#) will use WS-Discovery matching logic to verify that the scopes in the list match the scopes specified in *pProbeMessage*.

[in, optional] *pXAddrsList*

Pointer to a [WSD_URI_LIST](#) structure that represents the list of transport addresses supported by the publishing host. Each transport address string contains an address and port number which can be used for connection by a remote host. May be **NULL**.

[in, optional] *pHeaderAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the header.

[in, optional] *pReferenceParameterAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the reference parameter properties.

[in, optional] *pPolicyAny*

Not used.

[in, optional] *pEndpointReferenceAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the endpoint.

[in, optional] *pAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the message body.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: <i>pszId</i> is NULL. - The length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). - The length in characters of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). <i>pProbeMessage</i> is NULL.
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

MatchProbeEx should be called only when the discovery publisher has issued a [ProbeHandler](#) callback. *pProbeMessage* and *pMessageParameters* are passed directly from the callback into **MatchProbeEx**. The **ProbeHandler** also passes information required by the publisher to determine if the supplied Probe message matches and, if so, to issue a ProbeMatches response if appropriate.

MatchProbeEx sends ProbeMatches messages on all bound adapters and automatically issues message retransmissions when required by WS-Discovery.

The parameters referring to [WSDXML_ELEMENT](#) structures can be used to extend the contents of the ProbeMatches message being sent with custom information.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::MatchResolve method (wsddisco.h)

Article 10/13/2021

Determines whether a [Resolve](#) message matches the specified host and sends a WS-Discovery [ResolveMatches](#) message if the match is made.

Syntax

C++

```
HRESULT MatchResolve(  
    [in]          const WSD_SOAP_MESSAGE *pResolveMessage,  
    [in]          IWSDMessageParameters *pMessageParameters,  
    [in]          LPCWSTR                pszId,  
    [in]          ULONGLONG             ullMetadataVersion,  
    [in]          ULONGLONG             ullInstanceId,  
    [in]          ULONGLONG             ullMessageNumber,  
    [in, optional] LPCWSTR                pszSessionId,  
    [in, optional] const WSD_NAME_LIST    *pTypesList,  
    [in, optional] const WSD_URI_LIST     *pScopesList,  
    [in, optional] const WSD_URI_LIST     *pXAddrsList  
);
```

Parameters

[in] pResolveMessage

Pointer to a [WSD_SOAP_MESSAGE](#) structure that represents the Resolve message passed in to the notification sink's [ResolveHandler](#).

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) object that represents the transmission parameters passed in to the notification sink's [ResolveHandler](#).

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by http or https, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] ullMetadataVersion

Current metadata version.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in] ullInstanceId

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in] ullMessageNumber

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in, optional] pszSessionId

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the AppSequence block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#).

This parameter may be **NULL**.

[in, optional] pTypesList

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of types supported by the publishing host. May be **NULL**. If *pTypesList* is specified, **MatchResolve** will use WS-Discovery matching logic to verify that the types match those specified in *pResolveMessage*.

[in, optional] pScopesList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of matching scopes supported by the publishing host. The list contains hash values in string form. May be **NULL**. If *pScopesList* is specified, **MatchResolve** will use WS-Discovery matching logic to verify that the scopes match those specified in *pResolveMessage*.

[in, optional] *pXAddrsList*

Pointer to a [WSD_URI_LIST](#) structure that represents the list of transport addresses supported by the publishing host. Each transport address string contains an address and port number which can be used for connection by a remote host.*pXAddrsList* and *pXAddrsList->Element* may not be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: <i>pszId</i> is NULL. - The length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). - The length in characters of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). <i>pProbeMessage</i> is NULL.
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

MatchResolve should be called only when the discovery publisher has issued a [ResolveHandler](#) callback. *pResolveMessage* and *pMessageParameters* are passed directly from the callback into **MatchResolve**. The **ResolveHandler** also passes information required by the publisher to determine if the supplied Resolve message matches and, if so, to issue a **ResolveMatches** response if appropriate.

MatchResolve sends ResolveMatches messages on all bound adapters and automatically issues message retransmissions when required by WS-Discovery.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDDiscoveryPublisher::MatchResolveEx method (wsddisco.h)

Article 10/13/2021

Determines whether a [Resolve](#) message matches the specified host and sends a WS-Discovery [ResolveMatches](#) message with extended information if the match is made.

Syntax

C++

```
HRESULT MatchResolveEx(  
    [in]          const WSD_SOAP_MESSAGE *pResolveMessage,  
    [in]          IWSDMessageParameters *pMessageParameters,  
    [in]          LPCWSTR                pszId,  
    [in]          ULONGLONG             ullMetadataVersion,  
    [in]          ULONGLONG             ullInstanceId,  
    [in]          ULONGLONG             ullMessageNumber,  
    [in, optional] LPCWSTR                pszSessionId,  
    [in, optional] const WSD_NAME_LIST    *pTypesList,  
    [in, optional] const WSD_URI_LIST     *pScopesList,  
    [in, optional] const WSD_URI_LIST     *pXAddrsList,  
    [in, optional] const WSDXML_ELEMENT   *pHeaderAny,  
    [in, optional] const WSDXML_ELEMENT   *pReferenceParameterAny,  
    [in, optional] const WSDXML_ELEMENT   *pPolicyAny,  
    [in, optional] const WSDXML_ELEMENT   *pEndpointReferenceAny,  
    [in, optional] const WSDXML_ELEMENT   *pAny  
);
```

Parameters

[in] pResolveMessage

Pointer to a [WSD_SOAP_MESSAGE](#) structure that represents the Resolve message passed in to the notification sink's [ResolveHandler](#).

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) object that represents the transmission parameters passed in to the notification sink's [ResolveHandler](#).

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by `http` or `https`, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] `ullMetadataVersion`

Current metadata version.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in] `ullInstanceId`

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in] `ullMessageNumber`

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in, optional] `pszSessionId`

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the `AppSequence` block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#).

This parameter may be **NULL**.

[in, optional] `pTypesList`

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of types supported by the publishing host. May be **NULL**. If *pTypesList* is specified, **MatchResolveEx** will use WS-Discovery matching logic to verify that the types match those specified in *pResolveMessage*.

[in, optional] *pScopesList*

Pointer to a [WSD_URI_LIST](#) structure that represents the list of matching scopes supported by the publishing host. The list contains hash values in string form. May be **NULL**. If *pScopesList* is specified, **MatchResolveEx** will use WS-Discovery matching logic to verify that the scopes match those specified in *pResolveMessage*.

[in, optional] *pXAddrsList*

Pointer to a [WSD_URI_LIST](#) structure that represents the list of transport addresses supported by the publishing host. Each transport address string contains an address and port number which can be used for connection by a remote host. *pXAddrsList* and *pXAddrsList->Element* may not be **NULL**.

[in, optional] *pHeaderAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the header.

[in, optional] *pReferenceParameterAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the reference parameter properties.

[in, optional] *pPolicyAny*

Not used.

[in, optional] *pEndpointReferenceAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the endpoint.

[in, optional] *pAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the message body.

Return value

Possible return values include, but are not limited to, the following:

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: • <i>pszId</i> is NULL. • The length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). • The length in characters of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). • <i>pProbeMessage</i> is NULL.
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

MatchResolveEx should be only when the discovery publisher has issued a [ResolveHandler](#) callback. *pResolveMessage* and *pMessageParameters* are passed directly from the callback into **MatchResolveEx**. The **ResolveHandler** also passes information required by the publisher to determine if the supplied Resolve message matches and, if so, to issue a **ResolveMatches** response if appropriate.

MatchResolveEx sends **ResolveMatches** messages on all bound adapters and automatically issues message retransmissions when required by WS-Discovery.

The parameters referring to [WSDXML_ELEMENT](#) structures can be used to extend the contents of the **ResolveMatches** message being sent with custom information.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::Publish method (wsddisco.h)

Article 10/13/2021

Announces the presence of a network host by sending a [Hello](#) message.

Syntax

C++

```
HRESULT Publish(  
    [in] LPCWSTR pszId,  
    [in] ULONGLONG ullMetadataVersion,  
    [in] ULONGLONG ullInstanceId,  
    [in] ULONGLONG ullMessageNumber,  
    [in, optional] LPCWSTR pszSessionId,  
    [in, optional] const WSD_NAME_LIST *pTypesList,  
    [in, optional] const WSD_URI_LIST *pScopesList,  
    [in, optional] const WSD_URI_LIST *pXAddrsList  
);
```

Parameters

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by `http` or `https`, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] ullMetadataVersion

Current metadata version.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to `UINT_MAX` (4294967295).

[in] ullInstanceId

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in] ullMessageNumber

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in, optional] pszSessionId

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the AppSequence block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#).

This parameter may be **NULL**.

[in, optional] pTypesList

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of types supported by the publishing host. May be **NULL**.

[in, optional] pScopesList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of matching scopes supported by the publishing host. The list contains hash values in string form. May be **NULL**.

[in, optional] pXAddrsList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of transport addresses supported by the publishing host. Each transport address string contains an address and port number which can be used for connection by a remote host. May be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: • <i>pszId</i> is NULL. • The length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). • The length in characters of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
HRESULT_FROM_WIN32(ERROR_NO_CALLBACK_ACTIVE)	There is no registered notification sink. To attach a sink, call RegisterNotificationSink .
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

If successful, **Publish** will send a WS-Discovery Hello message to the local subnet with the provided information.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::PublishEx method (wsddisco.h)

Article 10/13/2021

Announces the presence of a network host by sending a [Hello](#) message with extended information.

Syntax

C++

```
HRESULT PublishEx(  
    [in] LPCWSTR pszId,  
    [in] ULONGLONG ullMetadataVersion,  
    [in] ULONGLONG ullInstanceId,  
    [in] ULONGLONG ullMessageNumber,  
    [in, optional] LPCWSTR pszSessionId,  
    [in, optional] const WSD_NAME_LIST *pTypesList,  
    [in, optional] const WSD_URI_LIST *pScopesList,  
    [in, optional] const WSD_URI_LIST *pXAddrsList,  
    [in, optional] const WSDXML_ELEMENT *pHeaderAny,  
    [in, optional] const WSDXML_ELEMENT *pReferenceParameterAny,  
    [in, optional] const WSDXML_ELEMENT *pPolicyAny,  
    [in, optional] const WSDXML_ELEMENT *pEndpointReferenceAny,  
    [in, optional] const WSDXML_ELEMENT *pAny  
);
```

Parameters

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by `http` or `https`, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] ullMetadataVersion

Current metadata version.

Note For compatibility with the WS-Discovery specification, this value must be less than or equal to `UINT_MAX` (4294967295).

[in] ullInstanceId

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the WS-Discovery specification, this value must be less than or equal to UINT_MAX (4294967295).

[in] ullMessageNumber

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the WS-Discovery specification, this value must be less than or equal to UINT_MAX (4294967295).

[in, optional] pszSessionId

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the AppSequence block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#).

This parameter may be **NULL**.

[in, optional] pTypesList

Pointer to a [WSD_NAME_LIST](#) structure that represents the list of types supported by the publishing host. May be **NULL**.

[in, optional] pScopesList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of matching scopes supported by the publishing host. The list contains hash values in string form. May be **NULL**.

[in, optional] pXAddrsList

Pointer to a [WSD_URI_LIST](#) structure that represents the list of transport addresses supported by the publishing host. Each transport address string contains an address and port number which can be used for connection by a remote host. May be **NULL**.

[in, optional] pHeaderAny

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the header.

[in, optional] pReferenceParameterAny

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the reference parameter properties.

[in, optional] pPolicyAny

Not used.

[in, optional] pEndpointReferenceAny

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the endpoint.

[in, optional] pAny

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the message body.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: <i>pszId</i> is NULL. - The length in characters of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). - The length in characters of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
HRESULT_FROM_WIN32(ERROR_NO_CALLBACK_ACTIVE)	There is no registered notification sink. To attach a sink, call RegisterNotificationSink .

E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

If successful, **PublishEx** will send a WS-Discovery Hello message to the local subnet with the provided information.

The parameters referring to [WSDXML_ELEMENT](#) structures can be used to extend the contents of the Hello message being sent with custom information.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::RegisterNotificationSink method (wsddisco.h)

Article02/22/2024

Attaches a callback notification sink to the discovery publisher.

Syntax

C++

```
HRESULT RegisterNotificationSink(  
    [in] IWSDiscoveryPublisherNotify *pSink  
);
```

Parameters

[in] pSink

Pointer to an [IWSDiscoveryPublisherNotify](#) object that represents the initialized interface to receive callback notifications. This parameter cannot be **NULL**.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	<i>pSink</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

The notification sink receives a callback whenever an inbound query is received. It is possible to register multiple notification sinks with a single publisher.

Note `RegisterNotificationSink` must be called at least once before any other `IWSDiscoveryPublisher` method is used.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::RegisterScopeMatchingRule method (wsddisco.h)

Article02/22/2024

Adds support for a custom scope matching rule.

Syntax

```
C++

HRESULT RegisterScopeMatchingRule(
    [in] IWSDScopeMatchingRule *pScopeMatchingRule
);
```

Parameters

[in] pScopeMatchingRule

Pointer to an [IWSDScopeMatchingRule](#) object that represents a custom scope matching rule.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	<i>pScopeMatchingRule</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

[RegisterScopeMatchingRule](#) allows custom scope matching rules to be defined and added to the existing set defined in the [WS-Discovery specification](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::SetAddressFamily method (wsddisco.h)

Article02/22/2024

Specifies the IP address family (IPv4, IPv6, or both) over which the host will be published.

Syntax

C++

```
HRESULT SetAddressFamily(  
    [in] DWORD dwAddressFamily  
);
```

Parameters

[in] dwAddressFamily

The address family over which the host will be published.

 Expand table

Value	Meaning
WSDAPI_ADDRESSFAMILY_IPV4	Publish the host over IPv4 addresses.
WSDAPI_ADDRESSFAMILY_IPV6	Publish the host over IPv6 addresses.
WSDAPI_ADDRESSFAMILY_IPV4 WSDAPI_ADDRESSFAMILY_IPV6	Publish the host over IPv4 and IPv6 addresses.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.

E_INVALIDARG	<i>dwAddressFamily</i> has a value other than WSDAPI_ADDRESSFAMILY_IPV4, WSDAPI_ADDRESSFAMILY_IPV6, or WSDAPI_ADDRESSFAMILY_IPV4 WSDAPI_ADDRESSFAMILY_IPV6.
STG_E_INVALIDFUNCTION	The address family has already been set for this publisher.
HRESULT_FROM_WIN32(WSAEsocktnosupport)	The system does not support the address family specified by <i>dwAddressFamily</i> .

Remarks

This method must be called before a notification sink is attached to the publisher.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::UnPublish method (wsddisco.h)

Article 10/13/2021

Announces the departure of a network host by sending a [Bye](#) message.

Syntax

C++

```
HRESULT UnPublish(  
    [in] LPCWSTR pszId,  
    [in] ULONGLONG ullInstanceId,  
    [in] ULONGLONG ullMessageNumber,  
    [in, optional] LPCWSTR pszSessionId,  
    [in, optional] const WSDXML_ELEMENT *pAny  
);
```

Parameters

[in] pszId

The logical or physical address of the device, which is used as the device endpoint address. A logical address is of the form `urn:uuid:{guid}`. A physical address can be a URI prefixed by `http` or `https`, or simply a URI prefixed by `uri`. Whenever possible, use a logical address.

[in] ullInstanceId

Identifier for the current instance of the device being published. This identifier must be incremented whenever the service is restarted. For more information about instance identifiers, see Appendix I of the [WS-Discovery specification](#).

Note For compatibility with the [WS-Discovery specification](#), this value must be less than or equal to UINT_MAX (4294967295).

[in] ullMessageNumber

Counter within the scope of the instance identifier for the current message. The message number must be incremented for each message.

Note For compatibility with the [WS-Discovery specification](#) [↗], this value must be less than or equal to UINT_MAX (4294967295).

[in, optional] *pszSessionId*

Unique identifier within the scope of the instance identifier for the current session. This parameter corresponds to the sequence identifier in the AppSequence block in the Probe message. For more information about sequence identifiers, see Appendix I of the [WS-Discovery specification](#) [↗].

This parameter may be **NULL**.

[in, optional] *pAny*

Pointer to a [WSDXML_ELEMENT](#) structure that contains an XML element to be inserted in the "ANY" section of the message body.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	One or more of the following conditions is true: <i>pszId</i> is NULL. - The length of <i>pszId</i> exceeds WSD_MAX_TEXT_LENGTH (8192). - The length of <i>pszSessionId</i> exceeds WSD_MAX_TEXT_LENGTH (8192).
E_ABORT	The publisher has not been started. Attaching a notification sink starts the publisher. To attach a sink, call RegisterNotificationSink .
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Remarks

If successful, **UnPublish** will send a WS-Discovery Bye message to the local subnet with the provided information.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::UnRegisterNotificationSink method (wsddisco.h)

Article02/22/2024

Detaches a callback notification sink from the discovery publisher.

Syntax

C++

```
HRESULT UnRegisterNotificationSink(  
    [in] IWSDiscoveryPublisherNotify *pSink  
);
```

Parameters

[in] pSink

Pointer to the [IWSDiscoveryPublisherNotify](#) interface that will stop receiving callback notifications.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	<i>pSink</i> is NULL .
E_FAIL	The method failed.

Remarks

Note `UnRegisterNotificationSink` must be called at least once for each notification sink previously attached to the discovery publisher.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisher::UnRegisterScopeMatchingRule method (wsddisco.h)

Article02/22/2024

Removes support for a custom scope matching rule.

Syntax

C++

```
HRESULT UnRegisterScopeMatchingRule(  
    [in] IWSDScopeMatchingRule *pScopeMatchingRule  
);
```

Parameters

[in] pScopeMatchingRule

Pointer to an [IWSDScopeMatchingRule](#) object that represents a custom scope matching rule.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The method completed successfully.
E_INVALIDARG	<i>pScopeMatchingRule</i> is NULL .

Remarks

UnRegisterScopeMatchingRule removes a previously associated custom scope matching rule.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisher](#)

IWSDiscoveryPublisherNotify interface (wsddisco.h)

Article02/22/2024

Is implemented by the client program to receive callback notifications from [IWSDiscoveryPublisher](#).

Inheritance

The `IWSDiscoveryPublisherNotify` interface inherits from the [IUnknown](#) interface. `IWSDiscoveryPublisherNotify` also has these types of members:

Methods

The `IWSDiscoveryPublisherNotify` interface has these methods.

 Expand table

IWSDiscoveryPublisherNotify::ProbeHandler
Is called when a Probe is received by the discovery publisher.
IWSDiscoveryPublisherNotify::ResolveHandler
Is called when a Resolve is received by the discovery publisher.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)

IWSDDiscoveryPublisherNotify::ProbeHandler method (wsddisco.h)

Article 02/22/2024

Is called when a [Probe](#) is received by the discovery publisher.

Syntax

C++

```
HRESULT ProbeHandler(  
    [in] const WSD_SOAP_MESSAGE *pSoap,  
    [in] IWSDMessageParameters *pMessageParameters  
);
```

Parameters

[in] pSoap

Pointer to a [WSD_SOAP_MESSAGE](#) structure that contains the Probe message received by the discovery publisher.

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) interface that contains transport information associated with the received message.

Return value

The return value is not meaningful. An implementer should return S_OK.

Remarks

ProbeHandler is called whenever a [Probe](#) is received by the discovery publisher. It is the responsibility of the callback interface to then call [MatchProbe](#) or [MatchProbeEx](#) with host information to determine whether or not the received Probe matches the host.

The body of the Probe message passed to *pSoap* can be cast to a [WSD_PROBE](#) structure.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisherNotify](#)

IWSDDiscoveryPublisherNotify::ResolveHandler method (wsddisco.h)

Article 02/22/2024

Is called when a [Resolve](#) is received by the discovery publisher.

Syntax

C++

```
HRESULT ResolveHandler(  
    [in] const WSD_SOAP_MESSAGE *pSoap,  
    [in] IWSDMessageParameters *pMessageParameters  
);
```

Parameters

[in] pSoap

Pointer to a [WSD_SOAP_MESSAGE](#) structure that contains the Resolve message received by the discovery publisher.

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) interface that contains transport information associated with the received message.

Return value

The return value is not meaningful. An implementer should return S_OK.

Remarks

ResolveHandler is called whenever a Resolve is received by the discovery publisher. It is the responsibility of the callback interface to then call [MatchResolve](#) or [MatchResolveEx](#) with host information to determine whether or not the received Resolve matches the host.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDiscoveryPublisherNotify](#)

IWSDMessageParameters interface (wsdbase.h)

Article02/22/2024

Use this interface to communicate message specific information up and down the protocol stack.

Inheritance

The IWSDMessageParameters interface inherits from the [IUnknown](#) interface. IWSDMessageParameters also has these types of members:

Methods

The IWSDMessageParameters interface has these methods.

 Expand table

IWSDMessageParameters::GetLocalAddress
Retrieves the generic address object representing the local address that received the message.
IWSDMessageParameters::GetLowerParameters
Retrieves message parameters from the layer below this layer in the protocol stack.
IWSDMessageParameters::GetRemoteAddress
Retrieves the generic address object representing the remote address from which the message was sent.
IWSDMessageParameters::SetLocalAddress
Sets a generic address object representing the source address that should send the message.
IWSDMessageParameters::SetRemoteAddress
Sets the generic address object representing the remote address to where the message is sent.

Remarks

In a request-response message pattern, the parameters also provide a way for the transport to determine where the response message for a given request should be sent. To enable this, the message parameters for a request must always be passed down the stack with the corresponding response.

Since message parameters can be packaged with a request or a response, and can be sent or received, the meaning of the local and remote address depends on the direction the message parameters.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

IWSDMessageParameters::GetLocalAddress method (wsdbase.h)

Article02/22/2024

Retrieves the generic address object representing the local address that received the message.

Syntax

```
C++

HRESULT GetLocalAddress(
    [out] IWSDAddress **ppAddress
);
```

Parameters

[out] ppAddress

An [IWSDAddress](#) interface that represents the local address that received the message.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppAddress</i> is NULL .

Remarks

The caller is responsible for releasing memory allocated to *ppAddress*.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDMessageParameters](#)

IWSDMessageParameters::GetLowerParameters method (wsdbase.h)

Article02/22/2024

Retrieves message parameters from the layer below this layer in the protocol stack.

Syntax

```
C++

HRESULT GetLowerParameters(
    [out] IWSDMessageParameters **ppTxParams
);
```

Parameters

[out] ppTxParams

An [IWSDMessageParameters](#) interface that you use to communicate message specific information up and down the protocol stack.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_NOTIMPL	The method was not implemented.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDMessageParameters](#)

IWSDMessageParameters::GetRemoteAddress method (wsdbase.h)

Article02/22/2024

Retrieves the generic address object representing the remote address from which the message was sent.

Syntax

C++

```
HRESULT GetRemoteAddress(  
    [out] IWSDAddress **ppAddress  
);
```

Parameters

[out] ppAddress

An [IWSDAddress](#) interface that represents the remote address from which the message was sent.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppAddress</i> is NULL .

Remarks

The caller is responsible for releasing memory allocated to *ppAddress*.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDMessageParameters](#)

IWSDMessageParameters::SetLocalAddress method (wsdbase.h)

Article02/22/2024

Sets a generic address object representing the source address that should send the message.

Syntax

C++

```
HRESULT SetLocalAddress(  
    [in] IWSDAddress *pAddress  
);
```

Parameters

[in] pAddress

An [IWSDAddress](#) interface that represents the source address that should send the message.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDMessageParameters](#)

IWSDMessageParameters::SetRemoteAddress method (wsdbase.h)

Article02/22/2024

Sets the generic address object representing the remote address to where the message is sent.

Syntax

C++

```
HRESULT SetRemoteAddress(  
    [in] IWSDAddress *pAddress  
);
```

Parameters

[in] pAddress

An [IWSDAddress](#) interface that represents the remote address to where the message is sent.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDMessageParameters](#)

IWSDMetadataExchange interface (wsdclient.h)

Article02/22/2024

Is the base class for other objects which access metadata.

Inheritance

The IWSDMetadataExchange interface inherits from the [IUnknown](#) interface. IWSDMetadataExchange also has these types of members:

Methods

The IWSDMetadataExchange interface has these methods.

 Expand table

IWSDMetadataExchange::GetMetadata
Retrieves metadata for an object.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

IWSDMetadataExchange::GetMetadata method (wsdclient.h)

Article02/22/2024

Retrieves metadata for an object.

Syntax

```
C++

HRESULT GetMetadata(
    [out] WSD_METADATA_SECTION_LIST **MetadataOut
);
```

Parameters

[out] MetadataOut

Pointer to a linked list of structures containing the requested metadata.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>MetadataOut</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDMetadataExchange](#)

IWSDOutboundAttachment interface (wsdattachment.h)

Article02/22/2024

Enables applications to send attachment data in a message using a MIME container.

Inheritance

The **IWSDOutboundAttachment** interface inherits from [IWSDAttachment](#).
IWSDOutboundAttachment also has these types of members:

Methods

The **IWSDOutboundAttachment** interface has these methods.

 Expand table

IWSDOutboundAttachment::Abort	
Aborts the transfer of data on the attachment MIME data stream.	
IWSDOutboundAttachment::Close	
Closes the current attachment MIME data stream. (IWSDOutboundAttachment.Close)	
IWSDOutboundAttachment::Write	
Sends attachment data to the remote host using a MIME container.	

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdattachment.h (include Wsdapi.h)

See also

[IWSDAttachment](#)

IWSDOutboundAttachment::Abort method (wsdattachment.h)

Article10/05/2021

Aborts the transfer of data on the attachment MIME data stream. When **Abort** is called, any pending data may be discarded.

Syntax

```
C++  
  
HRESULT Abort();
```

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
HRESULT_FROM_WIN32(ERROR_INVALID_OPERATION)	Abort was called before Write was called. You must call Write before terminating the attachment stream.

Remarks

The **Abort** method may be called when a **Close** or **Write** method call failed with the error STG_S_BLOCK.

Close must not be called once **Abort** has been called on an attachment stream.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDOutboundAttachment](#)

[IWSDOutboundAttachment::Close](#)

[IWSDOutboundAttachment::Write](#)

IWSDOutboundAttachment::Close method (wsdattachment.h)

Article07/27/2022

Closes the current attachment MIME data stream.

Syntax

```
C++  
  
HRESULT Close();
```

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully. All data in the attachment stream was successfully transferred.
HRESULT_FROM_WIN32(ERROR_INVALID_OPERATION)	Close was called before Write was called. You must call Write before closing the attachment stream.
STG_S_BLOCK	Internal buffers were not available. The data in the attachment stream was not successfully transferred.

Remarks

Close is used to indicate that the application has no more data to transmit in the current attachment stream. The return value can indicate an error in a previous **Write** operation or an issue closing the connection.

Close may block while waiting for a previous [Write](#) operation to complete. **Close** may block for up to 30 seconds (per HTTP transmission timeouts) while waiting for a previous **Write** operation to complete.

The **Close** method may return successfully after a failed **Close** attempt that returned **STG_S_BLOCK**. A subsequent success indicates that the internal buffers were freed for use after the initial failed attempt. When **STG_S_BLOCK** is received by an application, the application can either call **Close** again or terminate the data transfer using the [Abort](#) method.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDInboundAttachment](#)

[IWSDOutboundAttachment](#)

[IWSDOutboundAttachment::Abort](#)

[IWSDOutboundAttachment::Write](#)

IWSDOutboundAttachment::Write method (wsdattachment.h)

Article10/13/2021

Sends attachment data to the remote host using a MIME container.

Syntax

C++

```
HRESULT Write(  
    [in] const BYTE *pBuffer,  
    [in] DWORD      dwBytesToWrite,  
    [out] LPDWORD   pdwNumberOfBytesWritten  
);
```

Parameters

[in] pBuffer

Pointer to a buffer containing the output data. The application program is responsible for allocating and freeing this data buffer.

[in] dwBytesToWrite

Number of bytes to send to the remote host from *pBuffer*.

[out] pdwNumberOfBytesWritten

Pointer to a **DWORD** containing the number of bytes of data actually sent to the remote host.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pdwNumberOfBytesWritten</i> is NULL .

E_INVALIDARG	<i>pBuffer</i> is NULL .
HRESULT_FROM_WIN32(ERROR_INVALID_OPERATION)	The outbound attachment interface has not been initialized. Call WSDCreateOutboundAttachment to initialize the interface.
STG_S_BLOCK	Internal buffers were not available. The data was not accepted and queued for transmission.

Remarks

The **Write** method allows an application program to send arbitrary data to a remote host as a MIME-encapsulated message attachment. The first call to **Write** opens the outbound attachment stream and initiates transmission of the HTTP headers, envelope data, and the MIME-encoded application data. Subsequent calls to **Write** will send additional blocks of MIME-encoded application data until the application makes a call to [Close](#), which closes the attachment stream and finishes the transmission of the message envelope data and headers.

The **Write** operation may block under several conditions. On the initial operation, **Write** will block until the HTTP headers and XML content have been transmitted. When sending multiple attachments in a single message, the first call to **Write** on any attachment may block until any prior attachment streams have been completely transmitted. **Write** may block for up to 30 seconds (per HTTP transmission timeouts) if the remote host does not reply.

If an error occurs in establishing a connection or transmitting headers, **Write** will return the error code immediately. If a data transfer error occurs, the error may be delayed to a future call of **Write** or [Close](#).

The **Write** method may return successfully after a failed **Write** attempt that returned **STG_S_BLOCK**. A subsequent success indicates that the internal buffers were freed for use after the initial failed attempt. When **STG_S_BLOCK** is received by an application, the application can either resend the same data using the **Write** method or terminate the data transfer using the [Abort](#) method.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdattachment.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDInboundAttachment](#)

[IWSDOutboundAttachment](#)

[IWSDOutboundAttachment::Abort](#)

[IWSDOutboundAttachment::Close](#)

IWSDScopeMatchingRule interface (wsddisco.h)

Article02/22/2024

Is implemented by the client program to supply a custom scope matching rule which can be used to extend the standard scope matching rules defined in WS-Discovery.

Inheritance

The IWSDScopeMatchingRule interface inherits from the [IUnknown](#) interface. IWSDScopeMatchingRule also has these types of members:

Methods

The IWSDScopeMatchingRule interface has these methods.

 Expand table

IWSDScopeMatchingRule::GetScopeRule
Is called to return a URI defining the implemented scope matching rule.
IWSDScopeMatchingRule::MatchScopes
Is called to compare two scopes to determine if they match.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)

IWSDScopeMatchingRule::GetScopeRule method (wsddisco.h)

Article02/22/2024

Is called to return a URI defining the implemented scope matching rule.

Syntax

```
C++

HRESULT GetScopeRule(
    [out] LPCWSTR *ppszScopeMatchingRule
);
```

Parameters

[out] ppszScopeMatchingRule

Pointer to the scope matching rule. The implementer must allocate memory using [WSDAllocateLinkedMemory](#) and the caller must release memory using [WSDFreeLinkedMemory](#).

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

GetScopeRule should provide a copy of the URI for the scope matching rule this object represents. The copy will be released by the caller using [WSDFreeLinkedMemory](#).

ppszScopeMatchingRule should never be **NULL** or an empty string. To register for the **NULL** scope matching rule, register for the RFC2396 rule as defined in [WS-Discovery](#). Probe

messages containing a **NULL** MatchBy value will be converted to RFC2396 before **GetScopeRule** is called.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDScopeMatchingRule](#)

IWSDScopeMatchingRule::MatchScopes method (wsddisco.h)

Article02/22/2024

Is called to compare two scopes to determine if they match.

Syntax

```
C++

HRESULT MatchScopes(
    [in] LPCWSTR pszScope1,
    [in] LPCWSTR pszScope2,
    [out] BOOL *pfMatch
);
```

Parameters

[in] pszScope1

Pointer to the first scope matching rule.

[in] pszScope2

Pointer to the second scope matching rule.

[out] pfMatch

Set to **TRUE** if the scopes received via *pszScope1* and *pszScope2* match, **FALSE** otherwise.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

MatchScopes will be called on custom scope matching rules to determine whether or not the two scopes provided match. *pfMatch* should be assigned either **TRUE** or **FALSE** to indicate the match status.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsddisco.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDScopeMatchingRule](#)

IWSDServiceMessaging interface (wsdhost.h)

Article02/22/2024

The **IWSDServiceMessaging** interface is used by generated stub code to send faults or responses to incoming messages.

Inheritance

The **IWSDServiceMessaging** interface inherits from the [IUnknown](#) interface. **IWSDServiceMessaging** also has these types of members:

Methods

The **IWSDServiceMessaging** interface has these methods.

 Expand table

IWSDServiceMessaging::FaultRequest
Sends a fault matching a given request context.
IWSDServiceMessaging::SendResponse
Sends a response message matching a given request context.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)

IWSDServiceMessaging::FaultRequest method (wsdhost.h)

Article10/13/2021

Sends a fault matching a given request context. This method should be called only from [generated code](#).

Syntax

```
C++

HRESULT FaultRequest(
    [in]          WSD_SOAP_HEADER      *pRequestHeader,
    [in]          IWSDMessageParameters *pMessageParameters,
    [in, optional] WSD_SOAP_FAULT      *pFault
);
```

Parameters

[in] pRequestHeader

Pointer to a [WSD_SOAP_HEADER](#) structure that contains the SOAP header of the original request that caused the fault.

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) object that contains the message parameters for the original request that caused the fault.

[in, optional] pFault

Pointer to a [WSD_SOAP_FAULT](#) structure that describes the fault to serialize and send. If this parameter is omitted, a fault of type `wsa:EndpointUnavailable` will be sent.

Return value

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description

S_OK	Method succeeded.
E_INVALIDARG	<i>pRequestHeader</i> or <i>pMessageParameters</i> is NULL .
E_ABORT	The method could not be completed.
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceMessaging](#)

IWSDServiceMessaging::SendResponse method (wsdhost.h)

Article10/13/2021

Sends a response message matching a given request context. This method should be called only from [generated code](#).

Syntax

```
C++

HRESULT SendResponse(
    [in] void *pBody,
    [in] WSD_OPERATION *pOperation,
    [in] IWSDMessageParameters *pMessageParameters
);
```

Parameters

[in] pBody

Pointer to the message body to send in the response message.

[in] pOperation

Pointer to a [WSD_OPERATION](#) structure that contains the type of response to send.

[in] pMessageParameters

Pointer to an [IWSDMessageParameters](#) object that contains the message parameters from the original request message.

Return value

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	Method succeeded.

E_POINTER	<i>pOperation</i> or <i>pMessageParameters</i> is NULL .
E_ABORT	The method could not be completed.
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdhost.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceMessaging](#)

IWSDServiceProxy interface (wsdclient.h)

Article02/22/2024

Represents a remote WSD service for client applications and middleware.

Inheritance

The **IWSDServiceProxy** interface inherits from [IWSDMetadataExchange](#). **IWSDServiceProxy** also has these types of members:

Methods

The **IWSDServiceProxy** interface has these methods.

 Expand table

IWSDServiceProxy::BeginGetMetadata Initiates an asynchronous metadata exchange request with the remote service.
IWSDServiceProxy::EndGetMetadata Completes the asynchronous metadata exchange request and retrieves the service metadata from the response.
IWSDServiceProxy::GetEndpointProxy Gets the endpoint proxy for the device.
IWSDServiceProxy::GetServiceMetadata Retrieves the metadata for the IWSDServiceProxy object.
IWSDServiceProxy::SetEventingStatusCallback Sets or clears the eventing status callback.
IWSDServiceProxy::SubscribeToOperation Subscribes to a notification or solicit/response event.
IWSDServiceProxy::UnsubscribeToOperation Cancels a subscription to a notification or solicit/response event.

Remarks

Service proxy objects may reside on multiple endpoints. An endpoint more completely represents a URL (contains additional useful data). For example, one endpoint may support HTTP on IPv4 addresses and another may support HTTPS on IPv6 addresses. Since the same service lives on both endpoints, it is important that the service have underlying endpoint proxy objects, with each endpoint proxy corresponding to a single endpoint at which the service is available. The endpoint proxy takes care of simple messaging requests to the service, for example, sending one-way or two-way messages.

IWSDServiceProxy objects are employed to obtain service metadata, send messages to the service through a service proxy, subscribe to events on the service, and bind to proxies that provide type-specific semantics.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

See also

[IWSDMetadataExchange](#)

IWSDServiceProxy::BeginGetMetadata method (wsdclient.h)

Article02/22/2024

Initiates an asynchronous metadata exchange request with the remote service.

Syntax

C++

```
HRESULT BeginGetMetadata(  
    [out] IWSDAsyncResult **ppResult  
);
```

Parameters

[out] ppResult

An [IWSDAsyncResult](#) interface that you use to poll for the result, register a callback object, or configure an event to be signaled when the response is received.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppResult</i> is NULL .
E_ABORT	The method could not be completed.
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Remarks

Call [IWSDServiceProxy::EndGetMetadata](#) to complete the asynchronous operation and to retrieve the metadata.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxy::EndGetMetadata method (wsdclient.h)

Article 10/13/2021

Completes the asynchronous metadata exchange request and retrieves the service metadata from the response.

Syntax

C++

```
HRESULT EndGetMetadata(  
    [in] IWSDAsyncResult *pResult,  
    [out] WSD_METADATA_SECTION_LIST **ppMetadata  
);
```

Parameters

[in] pResult

An [IWSDAsyncResult](#) interface that represents the result of the request. Release this object when done.

[out] ppMetadata

Requested metadata. For details, see [WSD_METADATA_SECTION_LIST](#). Do not release this object.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Remarks

EndGetMetadata may block until the metadata retrieval operation has completed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxy::GetEndpointProxy method (wsdclient.h)

Article02/22/2024

Gets the endpoint proxy for the device.

Syntax

C++

```
HRESULT GetEndpointProxy(  
    [out] IWSDEndpointProxy **ppProxy  
);
```

Parameters

[out] ppProxy

Pointer to an [IWSDEndpointProxy](#) interface.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppProxy</i> is NULL.

Remarks

The endpoint proxy is provided if a fault was received for a prior request. The client can then call [IWSDEndpointProxy::GetFaultInfo](#) to determine the nature of the error.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxy::GetServiceMetadata method (wsdclient.h)

Article02/22/2024

Retrieves the metadata for the [IWSDServiceProxy](#) object.

Syntax

C++

```
HRESULT GetServiceMetadata(  
    [out] WSD_SERVICE_METADATA **ppServiceMetadata  
);
```

Parameters

[out] ppServiceMetadata

Reference to a [WSD_SERVICE_METADATA](#) structure that specifies service metadata. Do not release this object.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppServiceMetadata</i> is NULL .

Remarks

This metadata is also available as part of the metadata produced by [IWSDDeviceProxy::GetHostMetadata](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxy::SetEventingStatusCallback method (wsdclient.h)

Article02/22/2024

Sets or clears the eventing status callback.

Syntax

```
C++

HRESULT SetEventingStatusCallback(
    [in, optional] IWSDEventingStatus *pStatus
);
```

Parameters

[in, optional] pStatus

An [IWSDEventingStatus](#) interface that lets the client know of status changes in event subscriptions. If **NULL**, existing eventing status callbacks are cleared.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxy::SubscribeToOperation method (wsdclient.h)

Article 10/13/2021

Subscribes to a notification or solicit/response event.

Syntax

C++

```
HRESULT SubscribeToOperation(  
    [in] const WSD_OPERATION *pOperation,  
    [in] IUnknown *pUnknown,  
    [in] const WSDXML_ELEMENT *pAny,  
    [out] WSDXML_ELEMENT **ppAny  
);
```

Parameters

[in] pOperation

Reference to a [WSD_OPERATION](#) structure that specifies the operation to subscribe to.

[in] pUnknown

Anonymous data passed to a client eventing callback function. This data is used to associate a client object with the subscription.

[in] pAny

Extensible data to be added to the body of the subscription request. You can use the IWSDXML* interfaces to build the data. For details, see [WSDXML_ELEMENT](#).

[out] ppAny

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of event subscriptions. When done, call [WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). Do not release this object.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The proxy has already subscribed to the operation specified by <i>pOperation</i> .
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Remarks

This method is normally only called by generated proxy code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxy::UnsubscribeToOperation method (wsdclient.h)

Article02/22/2024

Cancels a subscription to a notification or solicit/response event.

Syntax

C++

```
HRESULT UnsubscribeToOperation(  
    [in] const WSD_OPERATION *pOperation  
);
```

Parameters

[in] pOperation

Reference to a [WSD_OPERATION](#) structure that specifies the operation subscribed to.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The proxy is not subscribed to the notification specified by <i>pOperation</i> .
E_POINTER	<i>pOperation</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxy](#)

IWSDServiceProxyEventing interface (wsdclient.h)

Article10/05/2021

Represents a remote WSD service for client applications and middleware. This interface allows for the implementation of multiple asynchronous operations.

Inheritance

The **IWSDServiceProxyEventing** interface inherits from **IWSDServiceProxy**. **IWSDServiceProxyEventing** also has these types of members:

Methods

The **IWSDServiceProxyEventing** interface has these methods.

 Expand table

IWSDServiceProxyEventing::BeginGetStatusForMultipleOperations

Begins an asynchronous operation that retrieves the current status.

IWSDServiceProxyEventing::BeginRenewMultipleOperations

Initializes an asynchronous operation that renews a collection of existing notification subscriptions by submitting a new duration.

IWSDServiceProxyEventing::BeginSubscribeToMultipleOperations

Initializes an asynchronous operation that subscribes to a collection of notifications or solicit/response events.

IWSDServiceProxyEventing::BeginUnsubscribeToMultipleOperations

Initializes an asynchronous cancelation request for a subscription to a collection of notifications or solicit/response events.

IWSDServiceProxyEventing::EndGetStatusForMultipleOperations

Completes an asynchronous operation that retrieves the current status.

IWSDServiceProxyEventing::EndRenewMultipleOperations

Completes an asynchronous operation that renews a collection of existing notification subscriptions by submitting a new duration.
IWSDServiceProxyEventing::EndSubscribeToMultipleOperations Completes an asynchronous operation that subscribes to a collection of notifications or solicit/response events.
IWSDServiceProxyEventing::EndUnsubscribeToMultipleOperations Completes an asynchronous cancellation request for a subscription to a collection of notifications or solicit/response events.
IWSDServiceProxyEventing::GetStatusForMultipleOperations Retrieves the current status.
IWSDServiceProxyEventing::RenewMultipleOperations Renews a collection of existing notification subscriptions by submitting a new duration.
IWSDServiceProxyEventing::SubscribeToMultipleOperations Subscribes to a collection of notifications or solicit/response events.
IWSDServiceProxyEventing::UnsubscribeToMultipleOperations Cancels a collection of subscriptions to notifications or solicit/response events.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)

IWSDServiceProxyEventing::BeginGetStatusForMultipleOperations method (wsdclient.h)

Article 10/13/2021

Begins an asynchronous operation that retrieves the current status for a collection of event subscriptions.

Syntax

C++

```
HRESULT BeginGetStatusForMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [in] const WSDXML_ELEMENT *pAny,  
    [in] IUnknown *pAsyncState,  
    [in] IWSDAsyncCallback *pAsyncCallback,  
    [out] IWSDAsyncResult **ppResult  
);
```

Parameters

[in] *pOperations*

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operation subscriptions to get status on.

[in] *dwOperationCount*

The number of elements in the array in *pOperations*.

[in] *pAny*

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[in] *pAsyncState*

Anonymous data passed to *pAsyncCallback* when the callback is called. This data is used to associate a client object with the pending operation. This parameter is optional.

[in] *pAsyncCallback*

Reference to an [IWSDAsyncCallback](#) object that performs the message callback status notifications. This parameter is optional.

[out] ppResult

Pointer to a pointer to an [IWSDAsyncResult](#) interface that will represent the result of the requests upon completion.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::BeginRenewMultipleOperations method (wsdclient.h)

Article 10/13/2021

Initializes an asynchronous operation that renews a collection of existing notification subscriptions by submitting a new duration.

Syntax

C++

```
HRESULT BeginRenewMultipleOperations(  
    [in] const WSD_OPERATION      *pOperations,  
    [in] DWORD                    dwOperationCount,  
    [in] const WSD_EVENTING_EXPIRES *pExpires,  
    [in] const WSDXML_ELEMENT      *pAny,  
    [in] IUnknown                 *pAsyncState,  
    [in] IWSDAsyncCallback         *pAsyncCallback,  
    [out] IWSDAsyncResult          **ppResult  
);
```

Parameters

[in] *pOperations*

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operation subscriptions to renew.

[in] *dwOperationCount*

The number of elements in the array in *pOperations*.

[in] *pExpires*

Pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies requested duration for the subscription.

[in] *pAny*

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[in] *pAsyncState*

Anonymous data passed to *pAsyncCallback* when the callback is called. This data is used to associate a client object with the pending operation. This parameter is optional.

[in] `pAsyncCallback`

Reference to an [IWSDAsyncCallback](#) object that performs the message callback status notifications. This parameter is optional.

[out] `ppResult`

Pointer to a pointer to an [IWSDAsyncResult](#) interface that will represent the result of the requests upon completion.

Return value

If this method succeeds, it returns `S_OK`. Otherwise, it returns an `HRESULT` error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::BeginSubscribeToMultipleOperations method (wsdclient.h)

Article 10/13/2021

Initializes an asynchronous operation that subscribes to a collection of notifications or solicit/response events.

Syntax

C++

```
HRESULT BeginSubscribeToMultipleOperations(  
    [in] const WSD_OPERATION      *pOperations,  
    [in] DWORD                    dwOperationCount,  
    [in] IUnknown                 *pUnknown,  
    [in] const WSD_EVENTING_EXPIRES *pExpires,  
    [in] const WSDXML_ELEMENT     *pAny,  
    [in] IUnknown                 *pAsyncState,  
    [in] IWSDAsyncCallback        *pAsyncCallback,  
    [out] IWSDAsyncResult         **ppResult  
);
```

Parameters

[in] `pOperations`

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operations of which to subscribe.

[in] `dwOperationCount`

The number of elements in the array in *pOperations*.

[in] `pUnknown`

Anonymous data passed to a client eventing callback function. This data is used to associate a client object with the subscription.

[in] `pExpires`

Pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the requested duration for the subscription.

[in] `pAny`

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[in] pAsyncState

Anonymous data passed to *pAsyncCallback* when the callback is called. This data is used to associate a client object with the pending operation. This parameter is optional.

[in] pAsyncCallback

Reference to an [IWSDAsyncCallback](#) object that performs the message callback status notifications. This parameter is optional.

[out] ppResult

Pointer to a pointer to an [IWSDAsyncResult](#) interface that will represent the result of the requests upon completion.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Remarks

This method is designed to be exclusively called by generated proxy code.

The method is asynchronous and will return immediately. The caller should subsequently call [EndSubscribeToMultipleOperations](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::BeginUnsubscribeToMultipleOperations method (wsdclient.h)

Article 02/22/2024

Initializes an asynchronous cancelation request for a subscription to a collection of notifications or solicit/response events.

Syntax

C++

```
HRESULT BeginUnsubscribeToMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [in] const WSDXML_ELEMENT *pAny,  
    [in] IUnknown *pAsyncState,  
    [in] IWSDAsyncCallback *pAsyncCallback,  
    [out] IWSDAsyncResult **ppResult  
);
```

Parameters

[in] *pOperations*

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operations to unsubscribe from.

[in] *dwOperationCount*

The number of elements in the array in *pOperations*.

[in] *pAny*

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[in] *pAsyncState*

Anonymous data passed to *pAsyncCallback* when the callback is called. This data is used to associate a client object with the pending operation. This parameter is optional.

[in] *pAsyncCallback*

Reference to an [IWSDAsyncCallback](#) object that performs the message callback status notifications. This parameter is optional.

[out] ppResult

Pointer to a pointer to an [IWSDAsyncResult](#) interface that will represent the result of the requests upon completion.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::EndGetStatusForMultipleOperations method (wsdclient.h)

Article 02/22/2024

Completes an asynchronous operation that retrieves the current status for a collection of event subscriptions.

Syntax

C++

```
HRESULT EndGetStatusForMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [in] IWSDAsyncResult *pResult,  
    [out] WSD_EVENTING_EXPIRES **ppExpires,  
    [out] WSDXML_ELEMENT **ppAny  
);
```

Parameters

[in] *pOperations*

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operation subscriptions to get status on.

[in] *dwOperationCount*

The number of elements in the array in *pOperations*.

[in] *pResult*

Pointer to an [IWSDAsyncResult](#) interface that represents the result of the requests upon completion.

[out] *ppExpires*

Pointer to a pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the duration of the subscription. Upon completion, call [WSDFreeLinkedMemory](#) to free the memory. This parameter is optional.

[out] *ppAny*

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of getstatu requests. When done, call [WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). This parameter is optional.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::EndRenewMultipleOperations method (wsdclient.h)

Article 02/22/2024

Completes an asynchronous operation that renews a collection of existing notification subscriptions by submitting a new duration.

Syntax

C++

```
HRESULT EndRenewMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [in] IWSDAsyncResult *pResult,  
    [out] WSD_EVENTING_EXPIRES **ppExpires,  
    [out] WSDXML_ELEMENT **ppAny  
);
```

Parameters

[in] *pOperations*

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operation subscriptions to renew.

[in] *dwOperationCount*

The number of elements in the array in *pOperations*.

[in] *pResult*

Pointer to an [IWSDAsyncResult](#) interface that represents the result of the requests upon completion.

[out] *ppExpires*

Pointer to a pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the duration of the subscription that was just renewed. Upon completion, call [WSDFreeLinkedMemory](#) to free the memory. This parameter is optional.

[out] *ppAny*

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of event subscriptions. When done, call [WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). This parameter is optional.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::EndSubscribeToMultipleOperations method (wsdclient.h)

Article 02/22/2024

Completes an asynchronous operation that subscribes to a collection of notifications or solicit/response events.

Syntax

C++

```
HRESULT EndSubscribeToMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [out] IWSDAsyncResult *pResult,  
    [out] WSD_EVENTING_EXPIRES **ppExpires,  
    [out] WSDXML_ELEMENT **ppAny  
);
```

Parameters

[in] *pOperations*

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the subscribed operations.

[in] *dwOperationCount*

The number of elements in the array in *pOperations*.

[out] *pResult*

Pointer to an [IWSDAsyncResult](#) interface that represents the result of the requests upon completion.

[out] *ppExpires*

Pointer to a pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the duration of the subscription. Upon completion, call [WSDFreeLinkedMemory](#) to free the memory. This parameter is optional.

[out] *ppAny*

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of event subscriptions. When done, call [WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). This parameter is optional.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Remarks

This method is designed to be exclusively called by generated proxy code.

The method is used to obtain the results from a previous asynchronous [BeginSubscribeToMultipleOperations](#) call.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::EndUnsubscribeToMultipleOperations method (wsdclient.h)

Article 02/22/2024

Completes an asynchronous cancellation request for a subscription to a collection of notifications or solicit/response events.

Syntax

C++

```
HRESULT EndUnsubscribeToMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [out] IWSDAsyncResult *pResult  
);
```

Parameters

[in] `pOperations`

Pointer to an array of references to [WSD_OPERATION](#) structures that specifies the operations from which to unsubscribe.

[in] `dwOperationCount`

The number of elements in the array in *pOperations*.

[out] `pResult`

Pointer to an [IWSDAsyncResult](#) interface that will represent the result of the requests upon completion.

Return value

If this method succeeds, it returns `S_OK`. Otherwise, it returns an `HRESULT` error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::GetStatusForMultipleOperations method (wsdclient.h)

Article 10/13/2021

Retrieves the current status for a collection of event subscriptions.

Syntax

C++

```
HRESULT GetStatusForMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [in] const WSDXML_ELEMENT *pAny,  
    [out] WSD_EVENTING_EXPIRES **ppExpires,  
    [out] WSDXML_ELEMENT **ppAny  
);
```

Parameters

[in] `pOperations`

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operation subscriptions to get status on.

[in] `dwOperationCount`

The number of elements in the array in *pOperations*.

[in] `pAny`

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[out] `ppExpires`

Pointer to a pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the duration of the subscription. Upon completion, call [WSDFreeLinkedMemory](#) to free the memory. This parameter is optional.

[out] `ppAny`

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of getstatus requests. When done, call

[WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). This parameter is optional.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::RenewMultipleOperations method (wsdclient.h)

Article 02/22/2024

Renews a collection of existing notification subscriptions by submitting a new duration.

Syntax

C++

```
HRESULT RenewMultipleOperations(  
    [in] const WSD_OPERATION      *pOperations,  
    [in] DWORD                    dwOperationCount,  
    [in] const WSD_EVENTING_EXPIRES *pExpires,  
    [in] const WSDXML_ELEMENT      *pAny,  
    [out] WSD_EVENTING_EXPIRES     **ppExpires,  
    [out] WSDXML_ELEMENT           **ppAny  
);
```

Parameters

[in] `pOperations`

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operation subscriptions to renew.

[in] `dwOperationCount`

The number of elements in the array in *pOperations*.

[in] `pExpires`

Pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies requested duration for the subscription.

[in] `pAny`

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[out] `ppExpires`

Pointer to a pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the duration of the subscription that was just renewed. Upon completion, call [WSDFreeLinkedMemory](#) to free the memory. This parameter is optional.

[out] ppAny

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of renew requests. When done, call [WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). This parameter is optional.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::SubscribeToMultipleOperations method (wsdclient.h)

Article 10/13/2021

Subscribes to a collection of notifications or solicit/response events.

Syntax

C++

```
HRESULT SubscribeToMultipleOperations(  
    [in] const WSD_OPERATION      *pOperations,  
    [in] DWORD                    dwOperationCount,  
    [in] IUnknown                  *pUnknown,  
    [in] const WSD_EVENTING_EXPIRES *pExpires,  
    [in] const WSDXML_ELEMENT      *pAny,  
    [out] WSD_EVENTING_EXPIRES     **ppExpires,  
    [out] WSDXML_ELEMENT           **ppAny  
);
```

Parameters

[in] `pOperations`

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operations of which to subscribe.

[in] `dwOperationCount`

The number of elements in the array in *pOperations*.

[in] `pUnknown`

Anonymous data passed to a client eventing callback function. This data is used to associate a client object with the subscription.

[in] `pExpires`

Pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies requested duration for the subscription.

[in] `pAny`

Pointer to extensible data to be added to the body of the request. This parameter is optional.

[out] ppExpires

Pointer to a pointer to a [WSD_EVENTING_EXPIRES](#) structure that specifies the duration of the subscription. Upon completion, call [WSDFreeLinkedMemory](#) to free the memory. This parameter is optional.

[out] ppAny

Extensible data that the remote device can add to the subscription response. This allows services to provide additional customization of event subscriptions. When done, call [WSDFreeLinkedMemory](#) to free the memory. For details, see [WSDXML_ELEMENT](#). This parameter is optional.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The proxy has already subscribed to the operation specified by <i>pOperation</i> .
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Remarks

This method is designed to be exclusively called by generated proxy code.

The method is synchronous and will return when the requests have completed or the expiration criteria have been satisfied.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDServiceProxyEventing::UnsubscribeToMultipleOperations method (wsdclient.h)

Article10/13/2021

Cancels a collection of subscriptions to notifications or solicit/response events.

Syntax

C++

```
HRESULT UnsubscribeToMultipleOperations(  
    [in] const WSD_OPERATION *pOperations,  
    [in] DWORD dwOperationCount,  
    [in] const WSDXML_ELEMENT *pAny  
);
```

Parameters

[in] pOperations

Pointer to an array of references to [WSD_OPERATION](#) structures that specify the operations to unsubscribe from.

[in] dwOperationCount

The number of elements in the array in *pOperations*.

[in] pAny

Pointer to extensible data to be added to the body of the request.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The proxy is not subscribed to the notification specified by

	<i>pOperation</i> .
E_POINTER	<i>pOperation</i> is NULL.

Remarks

This method is designed to be exclusively called by generated proxy code.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdclient.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDServiceProxyEventing](#)

IWSDSignatureProperty interface (wsdbase.h)

Article02/22/2024

Provides properties of signed messages.

Inheritance

The IWSDSignatureProperty interface inherits from the [IUnknown](#) interface. IWSDSignatureProperty also has these types of members:

Methods

The IWSDSignatureProperty interface has these methods.

 Expand table

IWSDSignatureProperty::GetSignature Gets the signature of a message.
IWSDSignatureProperty::GetSignedInfoHash Gets the hash of a message signature.
IWSDSignatureProperty::IsMessageSignatureTrusted Specifies if a message signature is trusted.
IWSDSignatureProperty::IsMessageSigned Specifies if a message is signed.

Remarks

An application can acquire this interface by calling the [QueryInterface](#) method of [IWSDiscoveredService](#).

IWSDSignatureProperty is useful to an application that wants to perform its own signature validation. By passing a **NULL** to the *pConfigParam* of [WSDCreateDiscoveryProvider2](#), the

internal signature validation is disabled and the provider can perform its own validation by examining these properties.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

IWSDSignatureProperty::GetKeyInfo method

Article 08/31/2015

Gets the subject key ID of the certificate for a signed message.

Syntax

C++

```
HRESULT GetKeyInfo(  
    [out]      BYTE  *pbKeyInfo,  
    [in, out]  DWORD *pdwKeyInfoSize  
);
```

Parameters

- *pbKeyInfo* [out]
A pointer to a buffer that will be filled with the subject key ID for the certificate that signed the message.
- *pdwKeyInfoSize* [in, out]
On input, the size of *pbKeyInfo* in bytes. On output, *pdwKeyInfoSize* contains the actual size of the buffer that was written.

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.
E_NOTAVAIL	The message is not signed.
HRESULT_FROM_WIN32(ERROR_MORE_DATA)	<i>pbKeyInfo</i> is not large enough to hold the information. <i>pdwKeyInfoSize</i> now specifies the required buffer size.

Remarks

The subject key ID is a hash of the certificate's public key and is used to uniquely identify a certificate.

If **NULL** is passed to *pbKeyInfo*, then **GetKeyInfo** will return the size of the buffer to allocate in the *pdwKeyInfoSize* parameter.

Requirements

 Expand table

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	Wsdbase.h (include Wsdapi.h)
IDL	Wsdbase.idl
DLL	Wsdapi.dll

See also

[IWSDSignatureProperty](#) 

IWSDSignatureProperty::GetSignature method (wsdbase.h)

Article02/22/2024

Gets the signature of a message.

Syntax

C++

```
HRESULT GetSignature(  
    [out]      BYTE  *pbSignature,  
    [in, out] DWORD *pdwSignatureSize  
);
```

Parameters

[out] pbSignature

A pointer to a buffer that will be filled with the signature of the message.

[in, out] pdwSignatureSize

On input, the size of *pbSignature* in bytes. On output, *pdwSignatureSize* contains the actual size of the buffer that was written.

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.
E_NOTAVAIL	The message is not signed.
HRESULT_FROM_WIN32(ERROR_MORE_DATA)	<i>pbSignature</i> is not large enough to hold the information. <i>pdwSignatureSize</i> now specifies the required buffer size.

Remarks

If **NULL** is passed to *pbSignature*, then **GetSignature** will return the size of the buffer to allocate in the *pdwSignatureSize* parameter.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDSignatureProperty](#)

IWSDSignatureProperty::GetSignedInfoHash method (wsdbase.h)

Article 10/13/2021

Gets the hash of a message signature.

Syntax

C++

```
HRESULT GetSignedInfoHash(  
    [out]     BYTE  *pbSignedInfoHash,  
    [in, out] DWORD *pdwHashSize  
);
```

Parameters

[out] pbSignedInfoHash

A pointer to a buffer that will be filled with the hash of the message signature.

[in, out] pdwHashSize

On input, the size of *pbSignedInfoHash* in bytes. On output, *pdwHashSize* contains the actual size of the buffer that was written.

Return value

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	Method succeeded.
E_NOTAVAIL	The message is not signed.
HRESULT_FROM_WIN32(ERROR_MORE_DATA)	<i>pbSignedInfoHash</i> is not large enough to hold the information. <i>pdwHashSize</i> now specifies the required buffer size.

Remarks

This is the hash of the <SignedInfo> node. The <SignedInfo> xml node contains the SHA1 hashes of the various parts of the signature that is to be included in the signature. The final XML message signature is computed by signing the hash of the <SignedInfo> node with the private key of the signing certificate.

If **NULL** is passed to *pbSignedInfoHash*, then **GetSignedInfoHash** will return the size of the buffer to allocate in the *pdwHashSize* parameter.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDSignatureProperty](#)

IWSDSignatureProperty::IsMessageSignatureTrusted method (wsdbase.h)

Article02/22/2024

Specifies if a message signature is trusted.

Syntax

C++

```
HRESULT IsMessageSignatureTrusted(  
    [out] BOOL *pbSignatureTrusted  
);
```

Parameters

[out] pbSignatureTrusted

A pointer to a boolean that specifies if a message signature is trusted.

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.

Remarks

A message is trusted if the signing certificate is among one of the certificates or in the certificate store passed down by the calling application in the [WSDCreateDiscoveryProvider2](#) call.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDSignatureProperty](#)

IWSDSignatureProperty::IsMessageSigned method (wsdbase.h)

Article02/22/2024

Specifies if a message is signed.

Syntax

C++

```
HRESULT IsMessageSigned(  
    [out] BOOL *pbSigned  
);
```

Parameters

[out] pbSigned

A pointer to a boolean that specifies if a message signature is signed.

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDSignatureProperty](#)

IWSDSSLClientCertificate interface (wsdbase.h)

Article02/22/2024

Retrieves the client SSL certificate.

Inheritance

The **IWSDSSLClientCertificate** interface inherits from the [IUnknown](#) interface. **IWSDSSLClientCertificate** also has these types of members:

Methods

The **IWSDSSLClientCertificate** interface has these methods.

 Expand table

IWSDSSLClientCertificate::GetClientCertificate Gets the client certificate.
IWSDSSLClientCertificate::GetMappedAccessToken Gets the mapped access token.

Remarks

An application can acquire this interface by calling the [QueryInterface](#) method of [IWSDHttpMessageParameters](#). If the connection did not arrive over SSL, the call to [QueryInterface](#) will return **E_NOINTERFACE**.

On the device host, the generated code calls the application's service method. This service method has access to the [IWSDHttpMessageParameters](#) interface through the [WSD_EVENT](#) structure. The **IWSDSSLClientCertificate** provides access to the client SSL certificate.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

IWSDSSLClientCertificate::GetClientCertificate method (wsdbase.h)

Article02/22/2024

Gets the client certificate.

Syntax

```
C++

HRESULT GetClientCertificate(
    [in, out] PCCERT_CONTEXT *ppCertContext
);
```

Parameters

[in, out] ppCertContext

A pointer to a [CERT_CONTEXT](#) structure that contains the client SSL certificate. Upon completion, the caller should free this memory by calling [CertFreeCertificateContext](#).

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.
E_NOTFOUND	A certificate is not available.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDSSLClientCertificate](#)

IWSDSSLClientCertificate::GetMappedAccessToken method (wsdbase.h)

Article 10/13/2021

Gets the mapped access token.

Syntax

C++

```
HRESULT GetMappedAccessToken(  
    [in, out] HANDLE *phToken  
);
```

Parameters

[in, out] phToken

A handle for the mapped access token. Upon completion, the caller must free the handle by calling [CloseHandle](#).

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.
S_FALSE	The token associated with the specified handle is not available.

Remarks

If the client certificate was successfully mapped to an operating system user account, then a valid access token for this user will be returned through *phToken*. This token can be used to impersonate the user. Internally, HTTP.sys will do the client certificate to user account mapping and return this information through the [HTTP_SSL_CLIENT_CERT_INFO](#) structure.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDSSLClientCertificate](#)

IWSDTransportAddress interface (wsdbase.h)

Article02/22/2024

Represents an IP-based transport address.

You should not create an instance of the **IWSDTransportAddress** interface. Instead, create an instance of either the [IWSDHttpAddress](#) or [IWSDUdpAddress](#) interface if an address object is required.

Inheritance

The **IWSDTransportAddress** interface inherits from [IWSDAddress](#). **IWSDTransportAddress** also has these types of members:

Methods

The **IWSDTransportAddress** interface has these methods.

 Expand table

IWSDTransportAddress::GetPort Gets the IP port number associated with this transport address.
IWSDTransportAddress::GetTransportAddress Gets a pointer to a string representation of the address object. (IWSDTransportAddress.GetTransportAddress)
IWSDTransportAddress::GetTransportAddressEx Gets a pointer to a string representation of the address object. (IWSDTransportAddress.GetTransportAddressEx)
IWSDTransportAddress::SetPort Sets only the IP port number for this transport address.
IWSDTransportAddress::SetTransportAddress Sets the string representation of the transport address.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

See also

[IWSDAddress](#)

IWSDTransportAddress::GetPort method (wsdbase.h)

Article02/22/2024

Gets the IP port number associated with this transport address.

Syntax

```
C++

HRESULT GetPort(
    [out] WORD *pwPort
);
```

Parameters

[out] pwPort

Port number associated with the address object.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pwPort</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDTransportAddress](#)

IWSDTransportAddress::GetTransportAddress method (wsdbase.h)

Article02/22/2024

Gets a pointer to a string representation of the address object. The format of the string varies, and is determined by the implementing interface (either [IWSDHttpAddress](#) or [IWSDUdpAddress](#)).

Syntax

```
C++

HRESULT GetTransportAddress(
    [out] LPCWSTR *ppszAddress
);
```

Parameters

[out] ppszAddress

String representation of the address object. Do not deallocate this pointer.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>ppszAddress</i> is NULL .
S_FALSE	The transport address has not yet been set. To set the transport address, call SetTransportAddress with a non- NULL address.

Remarks

The string returned by this method may contain an IPv4 or unbracketed IPv6 address such as "fe80::1". It may also contain a bracketed IPv6 address that includes the port such as "[fe80::1]:1234". The caller should parse the string carefully to account for both possibilities.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDTransportAddress](#)

IWSDTransportAddress::GetTransportAddressEx method (wsdbase.h)

Article02/22/2024

Gets a pointer to a string representation of the address object. The format of the string varies, and is determined by the implementing interface (either [IWSDHttpAddress](#) or [IWSDUdpAddress](#)).

Syntax

```
C++

HRESULT GetTransportAddressEx(
    [in]  BOOL    fSafe,
    [out] LPCWSTR *ppszAddress
);
```

Parameters

[in] fSafe

Specifies whether the scope identifier for an IPv6 address is included in the returned *ppszAddress* string. For example, if the address object represents an IPv6 link local address and *fSafe* is **FALSE**, then the IPv6 scope identifier will be included in the returned *ppszAddress* string.

If the address object represents an IPv4 address or a host name, this parameter is ignored.

[out] ppszAddress

String representation of the address object. Do not deallocate this pointer.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description

S_OK	Method completed successfully.
E_POINTER	<i>ppszAddress</i> is NULL .
S_FALSE	The transport address has not yet been set. To set the transport address, call SetTransportAddress with a non- NULL address.

Remarks

The string returned by this method may contain an IPv4 or unbracketed IPv6 address such as "fe80::1". It may also contain a bracketed IPv6 address that includes the port such as "[fe80::1]:1234". The caller should parse the string carefully to account for both possibilities.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDTransportAddress](#)

IWSDTransportAddress::SetPort method (wsdbase.h)

Article02/22/2024

Sets only the IP port number for this transport address.

Syntax

```
C++

HRESULT SetPort(
    [in] WORD wPort
);
```

Parameters

[in] wPort

The IP port number for the address object.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDTransportAddress](#)

IWSDTransportAddress::SetTransportAddress method (wsdbase.h)

Article02/22/2024

Sets the string representation of the transport address. The format of the string varies, and is determined by the implementing interface (either [IWSDHttpAddress](#) or [IWSDUdpAddress](#)).

Syntax

```
C++

HRESULT SetTransportAddress(
    [in] LPCWSTR pszAddress
);
```

Parameters

[in] pszAddress

String representation of the transport address.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	The length in characters of the address string pointed to by <i>ppszAddress</i> exceeds WSD_MAX_TEXT_LENGTH (8192), <i>ppszAddress</i> is NULL , or the format of <i>ppszAddress</i> was not recognized.
E_OUTOFMEMORY	Insufficient memory to complete the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDTransportAddress](#)

IWSDUdpAddress interface (wsdbase.h)

Article02/22/2024

Provides access to the individual components of a UDP address.

Inheritance

The **IWSDUdpAddress** interface inherits from [IWSDTransportAddress](#). **IWSDUdpAddress** also has these types of members:

Methods

The **IWSDUdpAddress** interface has these methods.

 Expand table

IWSDUdpAddress::GetAlias
Gets the alias for the discovery address.
IWSDUdpAddress::GetExclusive
Determines whether the socket is in exclusive mode.
IWSDUdpAddress::GetMessageType
Gets the message type for this UDP address configuration.
IWSDUdpAddress::GetSockaddr
Gets the socket address information.
IWSDUdpAddress::GetTTL
Gets the time-to-live (TTL) for all outbound packets using this address.
IWSDUdpAddress::SetAlias
Sets the alias for the discovery address.
IWSDUdpAddress::SetExclusive
Controls whether the socket is in exclusive mode.

[IWSDUdpAddress::SetMessageType](#)

Sets the message type for this UDP address configuration.

[IWSDUdpAddress::SetSockaddr](#)

Sets the socket address information.

[IWSDUdpAddress::SetTTL](#)

Sets the time-to-live (TTL) for all outbound packets using this address.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)

IWSDUdpAddress::GetAlias method (wsdbase.h)

Article02/22/2024

Gets the alias for the discovery address. This method is reserved for internal use and should not be called.

Syntax

C++

```
HRESULT GetAlias(  
    [out] GUID *pAlias  
);
```

Parameters

[out] pAlias

Pointer to the alias of the discovery address.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pAlias</i> is NULL.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::GetExclusive method (wsdbase.h)

Article02/22/2024

Determines whether the socket is in exclusive mode.

Syntax

```
C++  
  
HRESULT GetExclusive();
```

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	The socket is in exclusive mode.
S_FALSE	The socket is not in exclusive mode.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

IWSDUdpAddress::GetMessageType method (wsdbase.h)

Article02/22/2024

Gets the message type for this UDP address configuration. There are two types of messages; one-way messages, which do not require responses, and two-way messages, which do require responses.

Syntax

C++

```
HRESULT GetMessageType(  
    [out] WSDUdpMessageType *pMessageType  
);
```

Parameters

[out] pMessageType

Pointer to a [WSDUdpMessageType](#) value that specifies the message type used for this address configuration.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pMessageType</i> is NULL .

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::GetSockaddr method (wsdbase.h)

Article02/22/2024

Gets the socket address information.

Syntax

```
C++

HRESULT GetSockaddr(
    [out] SOCKADDR_STORAGE *pSockAddr
);
```

Parameters

[out] pSockAddr

Pointer to a [SOCKADDR_STORAGE](#) structure that contains the address information.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pSockAddr</i> is NULL .
E_FAIL	The method failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::GetTTL method (wsdbase.h)

Article02/22/2024

Gets the time-to-live (TTL) for all outbound packets using this address.

Syntax

C++

```
HRESULT GetTTL(  
    [out] DWORD *pdwTTL  
);
```

Parameters

[out] pdwTTL

Pointer to the TTL of outgoing UDP packets. Generally, the TTL represents the maximum number of hops before a packet is discarded. Some implementations interpret the TTL differently.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	The method succeeded.
E_POINTER	<i>pdwTTL</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::SetAlias method (wsdbase.h)

Article02/22/2024

Sets the alias for the discovery address. This method is reserved for internal use and should not be called.

Syntax

C++

```
HRESULT SetAlias(  
    [in] const GUID *pAlias  
);
```

Parameters

[in] pAlias

A pointer to the alias of the discovery address.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::SetExclusive method (wsdbase.h)

Article02/22/2024

Controls whether the socket is in exclusive mode.

Syntax

C++

```
HRESULT SetExclusive(  
    [in] BOOL fExclusive  
);
```

Parameters

[in] fExclusive

A value of **TRUE** indicates that the socket should be set to exclusive mode. A value of **FALSE** indicates that the socket should not be in exclusive mode.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method completed successfully.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::SetMessageType method (wsdbase.h)

Article02/22/2024

Sets the message type for this UDP address configuration. There are two types of messages: one-way messages, which do not require responses, and two-way messages, which do require responses.

Syntax

```
C++

HRESULT SetMessageType(
    [in] WSDUdpMessageType messageType
);
```

Parameters

[in] messageType

A [WSDUdpMessageType](#) value that specifies the message type used for this address configuration.

Return value

This method can return one of these values.

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::SetSockaddr method (wsdbase.h)

Article02/22/2024

Sets the socket address information.

Syntax

C++

```
HRESULT SetSockaddr(  
    [in] const SOCKADDR_STORAGE *pSockAddr  
);
```

Parameters

[in] pSockAddr

Pointer to a [SOCKADDR_STORAGE](#) structure.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pSockAddr</i> is NULL .
E_OUTOFMEMORY	Insufficient memory to complete the operation.
HRESULT_FROM_WIN32(WSAEINVAL)	The specified address is not a valid socket address, or no transport provider supports the indicated address family.
HRESULT_FROM_WIN32(WSANOTINITIALISED)	The Winsock 2 DLL has not been initialized. The application must first call WSAStartup to initialize Winsock 2.
HRESULT_FROM_WIN32(WSAENOBUFS)	No buffer space available.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpAddress::SetTTL method (wsdbase.h)

Article02/22/2024

Sets the time-to-live (TTL) for all outbound packets using this address.

Syntax

```
C++

HRESULT SetTTL(
    [in] DWORD dwTTL
);
```

Parameters

[in] dwTTL

The TTL of outgoing UDP packets. Generally, the TTL represents the maximum number of hops before a packet is discarded. Some implementations interpret the TTL differently.

Return value

Possible return values include, but are not limited to, the following.

 Expand table

Return code	Description
S_OK	Method succeeded.
E_INVALIDARG	<i>dwTTL</i> is greater than 255.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDUdpAddress](#)

IWSDUdpMessageParameters interface (wsdbase.h)

Article02/22/2024

Use this interface to specify how often WSD repeats the message transmission.

To get this interface from a UDP message sent during discovery, call the **QueryInterface** method of [IWSDMessageParameters](#) passing `__uuidof(IWSDUdpMessageParameters)` as the interface identifier.

You can also call [WSDCreateUdpMessageParameters](#) to retrieve this interface.

Inheritance

The **IWSDUdpMessageParameters** interface inherits from [IWSDMessageParameters](#). **IWSDUdpMessageParameters** also has these types of members:

Methods

The **IWSDUdpMessageParameters** interface has these methods.

 Expand table

IWSDUdpMessageParameters::GetRetransmitParams	
Retrieves the values that WSD uses to determine how often to repeat the message transmission.	
IWSDUdpMessageParameters::SetRetransmitParams	
Sets the values that WSD uses to determine how often to repeat the message transmission.	

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wsdbase.h

See also

[IWSDMessageParameters](#)

IWSDUdpMessageParameters::GetRetransmitParams method (wsdbase.h)

Article02/22/2024

Retrieves the values that WSD uses to determine how often to repeat the message transmission.

Syntax

C++

```
HRESULT GetRetransmitParams(  
    [out] WSDUdpRetransmitParams *pParams  
);
```

Parameters

[out] pParams

Pointer to a [WSDUdpRetransmitParams](#) structure. The structure contains values that determine how often WSD repeats the message transmission.

Return value

Possible return values include, but are not limited to, the following:

 Expand table

Return code	Description
S_OK	Method completed successfully.
E_POINTER	<i>pParams</i> is NULL .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h
DLL	Wsdapi.dll

See also

[IWSDUdpMessageParameters](#)

IWSDUdpMessageParameters::SetRetransmitParams method (wsdbase.h)

Article02/22/2024

Sets the values that WSD uses to determine how often to repeat the message transmission.

Syntax

C++

```
HRESULT SetRetransmitParams(  
    [in] const WSDUdpRetransmitParams *pParams  
);
```

Parameters

[in] pParams

Pointer to a [WSDUdpRetransmitParams](#) structure. The structure contains values that determine how often WSD repeats the message transmission.

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.

Remarks

If you do not specify these values, WSD sends the message only once.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdbase.h
DLL	Wsdapi.dll

See also

[IWSDUdpMessageParameters](#)

IWSDXMLContext interface (wsdxml.h)

Article02/22/2024

Is a collection of namespaces and types used in a WSDAPI stack.

Inheritance

The **IWSDXMLContext** interface inherits from the [IUnknown](#) interface. **IWSDXMLContext** also has these types of members:

Methods

The **IWSDXMLContext** interface has these methods.

 Expand table

IWSDXMLContext::AddNamespace
Creates an object that represents a namespace in an XML context.
IWSDXMLContext::AddNameToNamespace
Creates an object that represents a name in a namespace in an XML context.
IWSDXMLContext::SetNamespaces
Associates custom namespaces with the XML context object.
IWSDXMLContext::SetTypes
Associates custom message types with the XML context object.

Remarks

This interface is used by the XML parser and generator to store and access namespaces, names, and message schema information. Applications can call [AddNamespace](#) and [AddNameToNamespace](#) directly to add and access names in new or existing namespaces. Additionally, [generated code](#) will call [SetNamespaces](#) and [SetTypes](#) to ensure service layer data is properly set up in the XML context.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h (include Wsdapi.h)

IWSDXMLContext::AddNamespace method (wsdxml.h)

Article 10/13/2021

Creates an object that represents a namespace in an XML context. If the namespace already exists, no new namespace will be added, and the namespace object for the existing name will be returned.

Syntax

C++

```
HRESULT AddNamespace(  
    [in] LPCWSTR      pszUri,  
    [in] LPCWSTR      pszSuggestedPrefix,  
    [out] WSDXML_NAMESPACE **ppNamespace  
);
```

Parameters

[in] `pszUri`

The URI of the namespace.

[in] `pszSuggestedPrefix`

The namespace prefix to use when generating XML. If the namespace already exists, *pszSuggestedPrefix* will overwrite the prefix currently associated with the namespace. The XML context may assign a different namespace prefix. The prefix assigned by the XML context takes precedence over the suggested prefix. The **PreferredPrefix** member of the structure pointed to by *ppNamespace* contains the prefix assigned by the XML context.

[out] `ppNamespace`

Pointer to the address of the [WSDXML_NAMESPACE](#) structure that represents the namespace. You must deallocate *ppNamespace* by calling [WSDFreeLinkedMemory](#). This parameter is optional.

Return value

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description
S_OK	The method succeeded.
E_INVALIDARG	<i>pszUri</i> is NULL , the length in characters of the URI string exceeds WSD_MAX_TEXT_LENGTH (8192), <i>pszSuggestedPrefix</i> is NULL , or the length in characters of the prefix string exceeds WSD_MAX_TEXT_LENGTH (8192).
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Remarks

The returned [WSDXML_NAMESPACE](#) structure can be used to force an association between the namespace prefix (as specified by *pszSuggestedPrefix*) and the namespace URI (as specified by *pszUri*). Once this association is established, the XML generator will produce XML with the specified namespace prefix.

You can call **AddNamespace** to retrieve the [WSDXML_NAMESPACE](#) structure created when a namespace was automatically generated by a call to [AddNameToNamespace](#).

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDXMLContext](#)

IWSDXMLContext::AddNameToNamespace method (wsdxml.h)

Article 10/13/2021

Creates an object that represents a name in a namespace in an XML context. If the name already exists in the namespace, no new name will be added, and the name object for the existing name will be returned.

Syntax

C++

```
HRESULT AddNameToNamespace(  
    [in] LPCWSTR    pszUri,  
    [in] LPCWSTR    pszName,  
    [out] WSDXML_NAME **ppName  
);
```

Parameters

[in] *pszUri*

The URI of the XML namespace in which this name will be created. If this namespace does not already exist in the XML context, a new namespace structure will be generated automatically.

[in] *pszName*

The name to add to the namespace specified by *pszUri*.

[out] *ppName*

A [WSDXML_NAME](#) structure for the newly created name. You must deallocate *ppName* by calling [WSDFreeLinkedMemory](#). This parameter is optional.

Return value

Possible return values include, but are not limited to, the following.

[Expand table](#)

Return code	Description

S_OK	The method succeeded.
E_INVALIDARG	<i>pszUri</i> is NULL or the length in characters of the URI string exceeds WSD_MAX_TEXT_LENGTH (8192). <i>pszName</i> is NULL or the length in characters of the name string exceeds WSD_MAX_TEXT_LENGTH (8192).
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Remarks

AddNameToNamespace can be used when creating XML elements for extensible sections. Extensible sections are represented by the **any** element in a schema. The returned [WSDXML_NAME](#) structure pointed to by *ppName* can be used to specify the name associated with the extension content. When building a [WSDXML_ELEMENT](#) structure that represents extension content, use the returned **WSDXML_NAME** structure for the element's **Name** member.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDXMLContext](#)

IWSDXMLContext::SetNamespaces method (wsdxml.h)

Article02/22/2024

Associates custom namespaces with the XML context object.

This method should only be called by [generated code](#), and should not be called directly by a WSDAPI client. Instead, the code generator will provide wrappers that access this method properly.

Syntax

```
C++

HRESULT SetNamespaces(
    [in] const PCWSDXML_NAMESPACE *pNamespaces,
    [in] WORD wNamespacesCount,
    [in] BYTE bLayerNumber
);
```

Parameters

[in] pNamespaces

An array of [WSDXML_NAMESPACE](#) structures.

[in] wNamespacesCount

The number of namespaces in the *pNamespaces* array.

[in] bLayerNumber

The layer number associated with the [generated service code](#).

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description

S_OK	Method completed successfully.
E_INVALIDARG	<i>pNamespaces</i> is NULL or <i>bLayerNumber</i> is greater than or equal to WSD_XMLCONTEXT_NUM_LAYERS (16).
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDXMLContext](#)

IWSDXMLContext::SetTypes method (wsdxml.h)

Article 02/22/2024

Associates custom message types with the XML context object.

This method should only be called by [generated code](#), and should not be called directly by a WSDAPI client. Instead, the code generator will provide wrappers that access this method properly.

Syntax

C++

```
HRESULT SetTypes(  
    [in] const PCWSDXML_TYPE *pTypes,  
    [in] DWORD dwTypesCount,  
    [in] BYTE bLayerNumber  
);
```

Parameters

[in] pTypes

An array of [WSDXML_TYPE](#) structures that represent the set of messages for the [generated code](#).

[in] dwTypesCount

The number of types in the *pTypes* array.

[in] bLayerNumber

The layer number associated with the [generated service code](#).

Return value

Possible return values include, but are not limited to, the following:

[Expand table](#)

Return code	Description
S_OK	Method completed successfully.
E_INVALIDARG	<i>pTypes</i> is NULL or <i>bLayerNumber</i> is greater than or equal to WSD_XMLCONTEXT_NUM_LAYERS (16).
E_OUTOFMEMORY	Insufficient memory to complete the operation.
E_FAIL	The method failed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wsdxml.h (include Wsdapi.h)
DLL	Wsdapi.dll

See also

[IWSDXMLContext](#)

Web Services on Devices Structures

Article • 01/07/2021

The Web Services on Devices programming interface defines and uses the following structures:

- [REQUESTBODY_GetStatus](#)
- [REQUESTBODY_Renew](#)
- [REQUESTBODY_Subscribe](#)
- [REQUESTBODY_Unsubscribe](#)
- [RESPONSEBODY_GetMetadata](#)
- [RESPONSEBODY_GetStatus](#)
- [RESPONSEBODY_Renew](#)
- [RESPONSEBODY_Subscribe](#)
- [RESPONSEBODY_SubscriptionEnd](#)
- [WSD_APP_SEQUENCE](#)
- [WSD_BYE](#)
- [WSD_CONFIG_ADDRESSES](#)
- [WSD_CONFIG_PARAM](#)
- [WSD_DATETIME](#)
- [WSD_DURATION](#)
- [WSD_ENDPOINT_REFERENCE](#)
- [WSD_ENDPOINT_REFERENCE_LIST](#)
- [WSD_EVENT](#)
- [WSD_EVENTING_DELIVERY_MODE](#)
- [WSD_EVENTING_DELIVERY_MODE_PUSH](#)
- [WSD_EVENTING_EXPIRES](#)
- [WSD_EVENTING_FILTER](#)
- [WSD_EVENTING_FILTER_ACTION](#)
- [WSD_HANDLER_CONTEXT](#)
- [WSD_HEADER_RELATESTO](#)
- [WSD_HELLO](#)
- [WSD_HOST_METADATA](#)
- [WSD_LOCALIZED_STRING](#)
- [WSD_LOCALIZED_STRING_LIST](#)
- [WSD_METADATA_SECTION](#)
- [WSD_METADATA_SECTION_LIST](#)
- [WSD_NAME_LIST](#)
- [WSD_OPERATION](#)
- [WSD_PORT_TYPE](#)

- WSD_PROBE
 - WSD_PROBE_MATCH
 - WSD_PROBE_MATCH_LIST
 - WSD_PROBE_MATCHES
 - WSD_REFERENCE_PARAMETERS
 - WSD_REFERENCE_PROPERTIES
 - WSD_RELATIONSHIP_METADATA
 - WSD_RESOLVE
 - WSD_RESOLVE_MATCH
 - WSD_RESOLVE_MATCHES
 - WSD_SCOPES
 - WSD_SECURITY_CERT_VALIDATION
 - WSD_SECURITY_SIGNATURE_VALIDATION
 - WSD_SERVICE_METADATA
 - WSD_SERVICE_METADATA_LIST
 - WSD_SOAP_FAULT
 - WSD_SOAP_FAULT_CODE
 - WSD_SOAP_FAULT_REASON
 - WSD_SOAP_FAULT_SUBCODE
 - WSD_SOAP_HEADER
 - WSD_SOAP_MESSAGE
 - WSD_SYNCHRONOUS_RESPONSE_CONTEXT
 - WSD_THIS_DEVICE_METADATA
 - WSD_THIS_MODEL_METADATA
 - WSD_UNKNOWN_LOOKUP
 - WSD_URI_LIST
 - WSDUdpRetransmitParams
 - WSDXML_ATTRIBUTE
 - WSDXML_ELEMENT
 - WSDXML_ELEMENT_LIST
 - WSDXML_NAME
 - WSDXML_NAMESPACE
 - WSDXML_NODE
 - WSDXML_PREFIX_MAPPING
 - WSDXML_TEXT
 - WSDXML_TYPE
-

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

REQUESTBODY_GetStatus structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing GetStatus request message.

Syntax

C++

```
typedef struct {
    WSDXML_ELEMENT *Any;
} REQUESTBODY_GetStatus;
```

Members

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

Yes

No

REQUESTBODY_Renew structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing Renew request message.

Syntax

C++

```
typedef struct {
    WSD_EVENTING_EXPIRES *Expires;
    WSDXML_ELEMENT        *Any;
} REQUESTBODY_Renew;
```

Members

Expires

Reference to a [WSD_EVENTING_EXPIRES](#) structure that specifies when the renewed subscription will expire.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

REQUESTBODY_Subscribe structure (wsdtypes.h)

Article 02/22/2024

Represents a WS-Eventing Subscribe request message.

Syntax

C++

```
typedef struct {  
    WSD_ENDPOINT_REFERENCE      *EndTo;  
    WSD_EVENTING_DELIVERY_MODE *Delivery;  
    WSD_EVENTING_EXPIRES        *Expires;  
    WSD_EVENTING_FILTER          *Filter;  
    WSDXML_ELEMENT               *Any;  
} REQUESTBODY_Subscribe;
```

Members

EndTo

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that represents the endpoint reference of the event recipient.

Delivery

Reference to a [WSD_EVENTING_DELIVERY_MODE](#) structure that specifies the delivery mode. Only push delivery is supported.

Expires

Reference to a [WSD_EVENTING_EXPIRES](#) structure that specifies when the subscription will expire.

Filter

Reference to a [WSD_EVENTING_FILTER](#) structure that specifies a boolean expression used for event filtering.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

REQUESTBODY_Unsubscribe structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing Unsubscribe request message.

Syntax

C++

```
typedef struct {
    WSDXML_ELEMENT *any;
} REQUESTBODY_Unsubscribe;
```

Members

any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

Yes

No

RESPONSEBODY_GetMetadata structure (wsdtypes.h)

Article02/22/2024

Represents a WS-MetadataExchange GetMetadata response message.

Syntax

C++

```
typedef struct {
    WSD_METADATA_SECTION_LIST *Metadata;
} RESPONSEBODY_GetMetadata;
```

Members

Metadata

Reference to a [WSD_METADATA_SECTION_LIST](#) structure that contains a node in a single-linked list of metadata sections.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful? Yes No

RESPONSEBODY_GetStatus structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing GetStatus response message.

Syntax

C++

```
typedef struct {
    WSD_EVENTING_EXPIRES *expires;
    WSDXML_ELEMENT        *any;
} RESPONSEBODY_GetStatus;
```

Members

expires

Reference to a [WSD_EVENTING_EXPIRES](#) structure that specifies when the subscription expires.

any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

RESPONSEBODY_Renew structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing Renew response message.

Syntax

C++

```
typedef struct {
    WSD_EVENTING_EXPIRES *expires;
    WSDXML_ELEMENT        *any;
} RESPONSEBODY_Renew;
```

Members

expires

Reference to a [WSD_EVENTING_EXPIRES](#) structure that specifies when the subscription expires.

any

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) 

| [Get help at Microsoft Q&A](#)

RESPONSEBODY_Subscribe structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing Subscribe response message.

Syntax

C++

```
typedef struct {
    WSD_ENDPOINT_REFERENCE *SubscriptionManager;
    WSD_EVENTING_EXPIRES   *expires;
    WSDXML_ELEMENT         *any;
} RESPONSEBODY_Subscribe;
```

Members

SubscriptionManager

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that represents the endpoint reference of the subscription manager.

expires

Reference to a [WSD_EVENTING_EXPIRES](#) structure that specifies when the subscription expires.

any

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

RESPONSEBODY_SubscriptionEnd structure (wsdtypes.h)

Article02/22/2024

Represents a WS-Eventing SubscriptionEnd response message.

Syntax

C++

```
typedef struct {
    WSD_ENDPOINT_REFERENCE *SubscriptionManager;
    const WCHAR             *Status;
    WSD_LOCALIZED_STRING    *Reason;
    WSDXML_ELEMENT          *Any;
} RESPONSEBODY_SubscriptionEnd;
```

Members

SubscriptionManager

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that represents the endpoint reference of the subscription manager.

Status

A string that describes the reason the subscription ended.

 Expand table

Value	Meaning
http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown	The event source is shutting down.
http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling	The event source canceled the subscription for another reason.
http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure	The event source ended the subscription because

the delivery of
notifications failed.

Reason

Reference to a [WSD_LOCALIZED_STRING](#) that contains a human-readable explanation of the reason the subscription ended.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_APP_SEQUENCE structure (wsdtypes.h)

Article 02/22/2024

Represents application sequence information relating to WS-Discovery messages.

Syntax

C++

```
typedef struct _WSD_APP_SEQUENCE {  
    ULONGLONG InstanceId;  
    const WCHAR *SequenceId;  
    ULONGLONG MessageNumber;  
} WSD_APP_SEQUENCE;
```

Members

InstanceId

The instance identifier.

SequenceId

The sequence identifier.

MessageNumber

The message number.

Remarks

The application sequencing header block allows a receiver to maintain the sequence messages that contain this header block though they may have been received out of order. This allows proper sequencing of [Hello](#) and [Bye](#) messages from a target service.

The normative outline for the application sequence header block is:

syntax

```
<s:Envelope ...>
  <s:Header ...>
    <d:AppSequence InstanceId='xs:nonNegativeInteger'
[SequenceId='xs:anyURI']? MessageNumber='xs:nonNegativeInteger' ... />
  </s:Header>
  <s:Body ...> ...
</s:Body>
</s:Envelope>
```

The following describes normative constraints of this outline.

`/s:Envelope/s:Header/d:AppSequence/@InstanceId`

This setting must be incremented by a value of at least 1 each time the service has terminated, lost state, and been restored. An application can set this value by using a counter that is incremented each time a service is restarted. The restart time of the service is expressed as seconds elapsed since 12:00 a.m. January 1, 1970.

`/s:Envelope/s:Header/d:AppSequence/@SequenceId`

This setting identifies a sequence within the context of an instance identifier. If it is omitted, the implied value is the null sequence. The value in this setting must be unique within `./@InstanceId`.

`/s:Envelope/s:Header/d:AppSequence/@MessageNumber`

This setting identifies a message within the context of a sequence identifier and an instance identifier. must be incremented by a value of at least 1 for each message sent. Retransmission of this message at the transport level must maintain this value.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

See also

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_BYE structure (wsdtypes.h)

Article02/22/2024

Represents a [Bye](#) message.

Syntax

C++

```
typedef struct _WSD_BYE {
    WSD_ENDPOINT_REFERENCE *EndpointReference;
    WSDXML_ELEMENT         *Any;
} WSD_BYE;
```

Members

EndpointReference

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies either the sending or receiving endpoint of the Bye message.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Bye Message](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_CONFIG_ADDRESSES structure (wsdbase.h)

Article02/22/2024

Information about specific addresses that a host should listen on.

Syntax

C++

```
typedef struct _WSD_CONFIG_ADDRESSES {
    IWSDAddress **addresses;
    DWORD        dwAddressCount;
} WSD_CONFIG_ADDRESSES, *PWSD_CONFIG_ADDRESSES;
```

Members

addresses

An array of pointers to [IWSDAddress](#) interfaces.

If *pszLocalId* contains a logical address, the resulting behavior is a mapping between the logical address and a specific set of physical addresses (instead of a mapping between the logical address and a default physical address).

dwAddressCount

The number of items in the **addresses** array.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	wsdbase.h (include Windows.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_CONFIG_PARAM structure (wsdbase.h)

Article02/22/2024

Represents configuration parameters for creating WSDAPI objects.

Syntax

C++

```
typedef struct _WSD_CONFIG_PARAM {
    WSD_CONFIG_PARAM_TYPE configParamType;
    PVOID                  pConfigData;
    DWORD                  dwConfigDataSize;
} WSD_CONFIG_PARAM, *PWSD_CONFIG_PARAM;
```

Members

configParamType

A WSD_CONFIG_PARAM_TYPE value that indicates the type configuration data contained in this structure.

pConfigData

A pointer to a single configuration data structure. The configParamType member specifies the type of data passed in.

dwConfigDataSize

The size of the configuration data in pConfigData.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]

Requirement	Value
Header	wsdbase.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_DATETIME structure (wsdxml.h)

Article 02/22/2024

Represents a timestamp.

Syntax

C++

```
typedef struct _WSD_DATETIME {  
    BOOL    isPositive;  
    ULONG   year;  
    UCHAR   month;  
    UCHAR   day;  
    UCHAR   hour;  
    UCHAR   minute;  
    UCHAR   second;  
    UINT    millisecond;  
    BOOL    TZIsLocal;  
    BOOL    TZIsPositive;  
    UCHAR   TZHour;  
    UCHAR   TZMinute;  
} WSD_DATETIME;
```

Members

`isPositive`

TRUE if *year* value is positive.

`year`

Year value (for example, 2005). This number is a value between 0 and max(ULONG).

`month`

One-based month value (1 = January, through 12 = December).

`day`

One-based day of the month value (1-31).

`hour`

Zero-based hour value (0 through 23). *hour*=24 is only allowed if both *minute* and *second* are 0.

minute

Zero-based minute value (0 through 59).

second

Zero-based second value (0 through 59).

millisecond

Millisecond value (0-999). When this structure is converted to XML, the millisecond value is expressed as a fraction of a second in decimal form. For example, if **millisecond** has a value of 9, then the XML output will be 0.009.

TZIsLocal

TRUE if date and time are based on the local time zone, **FALSE** if UTC + offset.

TZIsPositive

TRUE if time zone offset specified by *TZHour* and *TZMinute* is positive relative to UTC, **FALSE** if offset is negative. Not valid if *TZIsLocal* is **TRUE**.

TZHour

Time zone offset relative to UTC (0-13). *TZhour*=14 is allowed if *TZMinute* is 0. Not valid if *TZIsLocal* is **TRUE**.

TZMinute

Time zone offset relative to UTC (0-59). Not valid if *TZIsLocal* is **TRUE**.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxml.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_DURATION structure (wsdxml.h)

Article 02/22/2024

Represents a length of time.

Syntax

C++

```
typedef struct _WSD_DURATION {  
    BOOL    isPositive;  
    ULONG   year;  
    ULONG   month;  
    ULONG   day;  
    ULONG   hour;  
    ULONG   minute;  
    ULONG   second;  
    ULONG   millisecond;  
} WSD_DURATION;
```

Members

`isPositive`

This parameter is **TRUE** if the entire duration is positive.

`year`

The year value. This number is a value between 0 and max(ULONG).

`month`

The month value. This number is a value between 0 and max(ULONG).

`day`

The day value. This number is a value between 0 and max(ULONG).

`hour`

The hour value. This number is a value between 0 and max(ULONG).

`minute`

The minute value. This number is a value between 0 and max(ULONG).

`second`

The second value. This number is a value between 0 and max(ULONG).

`millisecond`

The millisecond value (0-999).

Remarks

If any numeric member has a value of 0, then the member and its value is not included in the XML output when the structure is converted to XML.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxml.h (include Wsdapi.h)

Feedback

Was this page helpful?

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WSD_ENDPOINT_REFERENCE structure (wsdtypes.h)

Article 02/22/2024

Represents a WS-Addressing endpoint reference.

Syntax

C++

```
typedef struct _WSD_ENDPOINT_REFERENCE {  
    const WCHAR                *Address;  
    WSD_REFERENCE_PROPERTIES ReferenceProperties;  
    WSD_REFERENCE_PARAMETERS ReferenceParameters;  
    WSDXML_NAME                *PortType;  
    WSDXML_NAME                *ServiceName;  
    WSDXML_ELEMENT             *Any;  
} WSD_ENDPOINT_REFERENCE;
```

Members

Address

The endpoint address.

ReferenceProperties

[WSD_REFERENCE_PROPERTIES](#) structure that specifies additional data used to uniquely identify the endpoint.

ReferenceParameters

[WSD_REFERENCE_PARAMETERS](#) structure that specifies additional opaque data used by the endpoint.

PortType

Reference to a [WSDXML_NAME](#) structure that specifies the port type of the service at the referenced endpoint.

ServiceName

Reference to a [WSDXML_NAME](#) structure that specifies the service name of the service at the referenced endpoint.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h

Feedback

Was this page helpful?  Yes  No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_ENDPOINT_REFERENCE_LIST structure (wsdtypes.h)

Article02/22/2024

Represents a node in a single-linked list of [WSD_ENDPOINT_REFERENCE](#) structures.

Syntax

C++

```
typedef struct _WSD_ENDPOINT_REFERENCE_LIST {
    WSD_ENDPOINT_REFERENCE_LIST *Next;
    WSD_ENDPOINT_REFERENCE      *Element;
} WSD_ENDPOINT_REFERENCE_LIST;
```

Members

Next

Reference to the next node in the linked list of [WSD_ENDPOINT_REFERENCE_LIST](#) structures.

Element

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that contains the endpoint referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_EVENT structure (wsdtypes.h)

Article 10/05/2021

Provides an internal representation of a SOAP message.

Syntax

C++

```
typedef struct _WSD_EVENT {  
    HRESULT                Hr;  
    DWORD                  EventType;  
    WCHAR                  *DispatchTag;  
    WSD_HANDLER_CONTEXT    HandlerContext;  
    WSD_SOAP_MESSAGE        *Soap;  
    WSD_OPERATION           *Operation;  
    struct IWSDMessageParameters *MessageParameters;  
} WSD_EVENT;
```

Members

Hr

The result code of the event.

EventType

The event type.

DispatchTag

Pointer to the protocol string when dispatch by tags is required.

HandlerContext

Reference to a [WSD_HANDLER_CONTEXT](#) structure that specifies the handler context.

Soap

Reference to a [WSD_SOAP_MESSAGE](#) structure that describes the event.

Operation

Reference to a [WSD_OPERATION](#) structure that specifies the operation performed.

Message transmission parameters.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

WSD_EVENTING_DELIVERY_MODE structure (wsdtypes.h)

Article02/22/2024

Represents the delivery mode used in a WS-Eventing Subscribe message.

Syntax

C++

```
typedef struct _WSD_EVENTING_DELIVERY_MODE {
    const WCHAR *Mode;
    WSD_EVENTING_DELIVERY_MODE_PUSH *Push;
    void *Data;
} WSD_EVENTING_DELIVERY_MODE;
```

Members

Mode

Specifies the delivery mode for event delivery.

 Expand table

Value	Meaning
http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push	Push mode delivery is used.

Push

Data

A reference to the endpoint used for event delivery.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_EVENTING_DELIVERY_MODE_PUSH structure (wsdtypes.h)

Article02/22/2024

Represents the endpoint reference used for push delivery of events in a WS-Eventing Subscribe message.

Syntax

C++

```
typedef struct _WSD_EVENTING_DELIVERY_MODE_PUSH {
    WSD_ENDPOINT_REFERENCE *NotifyTo;
} WSD_EVENTING_DELIVERY_MODE_PUSH;
```

Members

NotifyTo

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies the endpoint reference to which notifications should be sent.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

WSD_EVENTING_EXPIRES structure (wsdtypes.h)

Article02/22/2024

Represents the expiration time of a WS-Eventing message.

Syntax

C++

```
typedef struct _WSD_EVENTING_EXPIRES {
    WSD_DURATION *Duration;
    WSD_DATETIME *DateTime;
} WSD_EVENTING_EXPIRES;
```

Members

Duration

Reference to a [WSD_DURATION](#) structure that specifies the length of time a request or response is valid.

DateTime

Reference to a [WSD_DATETIME](#) structure that specifies the time that the request or response expires.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_EVENTING_FILTER structure (wsdtypes.h)

Article02/22/2024

Represents an event filter used in WS-Eventing Subscribe messages.

Syntax

C++

```
typedef struct _WSD_EVENTING_FILTER {
    const WCHAR          *Dialect;
    WSD_EVENTING_FILTER_ACTION *FilterAction;
    void                 *Data;
} WSD_EVENTING_FILTER;
```

Members

Dialect

Specifies the language or dialect use to represent the boolean expression used by the filter.

 Expand table

Value	Meaning
http://schemas.xmlsoap.org/ws/2006/02/devprof/Action	The boolean expression uses the Action filter dialect.

FilterAction

Data

A reference to the expression used for filtering.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_EVENTING_FILTER_ACTION structure (wsdtypes.h)

Article02/22/2024

Represents a boolean expression used for filtering events

Syntax

C++

```
typedef struct _WSD_EVENTING_FILTER_ACTION {
    WSD_URI_LIST *Actions;
} WSD_EVENTING_FILTER_ACTION;
```

Members

Actions

Reference to a [WSD_URI_LIST](#) structure that specifies the URIs used for filtering notifications.

Remarks

For more information about the evaluation of action filters, see Section 6.1.1, Filtering, in the [Device Profile for Web Services](#) [↗](#) specification.

Requirements

[↗](#) Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_HANDLER_CONTEXT structure (wsdtypes.h)

Article 02/22/2024

Specifies the context for handling incoming messages.

Syntax

C++

```
typedef struct _WSD_HANDLER_CONTEXT {  
    PWSOAP_MESSAGE_HANDLER Handler;  
    void *PVoid;  
    IUnknown *Unknown;  
} WSD_HANDLER_CONTEXT;
```

Members

Handler

[PWSOAP_MESSAGE_HANDLER](#) function that specifies the incoming message handler.

PVoid

The value supplied by the *pVoidContext* parameter of the `IWSDSession::AddPort`, `IWSDSession::RegisterForIncomingRequests`, or `IWSDSession::RegisterForIncomingResponse` methods.

Unknown

The value supplied by the *unknownContext* parameter of the `IWSDSession::AddPort`, `IWSDSession::RegisterForIncomingRequests`, or `IWSDSession::RegisterForIncomingResponse` methods.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_HEADER_RELATESTO structure (wsdtypes.h)

Article02/22/2024

Represents a RelatesTo SOAP envelope header block, as specified by the WS-Addressing specification.

Syntax

C++

```
typedef struct _WSD_HEADER_RELATESTO {
    WSDXML_NAME *RelationshipType;
    const WCHAR *MessageID;
} WSD_HEADER_RELATESTO;
```

Members

RelationshipType

Reference to a [WSDXML_NAME](#) structure that contains the relationship type as a qualified name.

MessageID

The identifier of the related message.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_HELLO structure (wsdypes.h)

Article 10/05/2021

Represents a [Hello](#) message.

Syntax

C++

```
typedef struct _WSD_HELLO {  
    WSD_ENDPOINT_REFERENCE *EndpointReference;  
    WSD_NAME_LIST           *Types;  
    WSD_SCOPES              *Scopes;  
    WSD_URI_LIST            *XAddrs;  
    ULONGLONG               MetadataVersion;  
    WSDXML_ELEMENT          *Any;  
} WSD_HELLO;
```

Members

EndpointReference

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies the endpoint announcing the Hello message.

Types

Reference to a [WSD_NAME_LIST](#) structure that contains a list of WS-Discovery Types.

Scopes

Reference to a [WSD_SCOPES](#) structure that contains a list of WS-Discovery Scopes.

XAddrs

Reference to a [WSD_URI_LIST](#) structure that contains a list of WS-Discovery XAddrs.

MetadataVersion

The metadata version of this message.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Hello Message](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD_HOST_METADATA structure (wsdtypes.h)

Article02/22/2024

Provides metadata for all services hosted by a device.

Syntax

C++

```
typedef struct _WSD_HOST_METADATA {
    WSD_SERVICE_METADATA      *Host;
    WSD_SERVICE_METADATA_LIST *Hosted;
} WSD_HOST_METADATA;
```

Members

Host

Reference to a [WSD_SERVICE_METADATA](#) structure that describes the parent service or the device.

Hosted

Reference to a [WSD_SERVICE_METADATA_LIST](#) structure that represents the singly linked list of services hosted by the parent service.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_LOCALIZED_STRING structure (wsdtypes.h)

Article02/22/2024

Represents a single localized string.

Syntax

C++

```
typedef struct _WSD_LOCALIZED_STRING {
    const WCHAR *lang;
    const WCHAR *String;
} WSD_LOCALIZED_STRING;
```

Members

lang

The standard language code used for localization. Valid language codes are specified in [RFC 1766](#).

String

The string data in the localized language.

Remarks

[RFC 1766](#) extends [ISO-639](#). Dialect extensions to the [ISO-639](#) codes are used for the *lang* member. For example, "en-US" is used to indicate a string localized for the USA/English dialect.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_LOCALIZED_STRING_LIST structure (wsdtypes.h)

Article02/22/2024

Represents a node in a single-linked list of localized strings.

Syntax

C++

```
typedef struct _WSD_LOCALIZED_STRING_LIST {
    WSD_LOCALIZED_STRING_LIST *Next;
    WSD_LOCALIZED_STRING      *Element;
} WSD_LOCALIZED_STRING_LIST;
```

Members

Next

Reference to the next node in the linked list of **WSD_LOCALIZED_STRING_LIST** structures.

Element

The localized string referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) 

| [Get help at Microsoft Q&A](#)

WSD_METADATA_SECTION structure (wsdtypes.h)

Article02/22/2024

Represents a section of metadata in a generic form.

Note Only one of the **Data**, **MetadataReference**, or **Location** members should be specified.

Syntax

```
C++

typedef struct _WSD_METADATA_SECTION {
    const WCHAR          *Dialect;
    const WCHAR          *Identifier;
    void                 *Data;
    WSD_ENDPOINT_REFERENCE *MetadataReference;
    const WCHAR          *Location;
    WSDXML_ELEMENT       *Any;
} WSD_METADATA_SECTION;
```

Members

Dialect

The format and version of the metadata section.

 Expand table

Value	Meaning
http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel	The metadata section contains model-specific information relating to the device. If the Data member is specified, then its type is WSD_THIS_MODEL_METADATA .

<code>http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice</code>	The metadata section contains metadata that is unique to a specific device. If the Data member is specified, then its type is WSD_THIS_DEVICE_METADATA .
<code>http://schemas.xmlsoap.org/ws/2006/02/devprof/Relationship</code>	The metadata section contains metadata about the relationship between two or more services. If the Data member is specified, then its type is WSD_RELATIONSHIP_METADATA .

Identifier

The dialect-specific identifier for the scope/domain/namespace of the metadata section.

Data

Reference to a binary representation of the metadata. The type of metadata is specified by **Dialect**. This member is ignored if **Dialect** does not have a value of

`http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel`,

`http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice`, or

`http://schemas.xmlsoap.org/ws/2006/02/devprof/Relationship`.

MetadataReference

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure used identify the endpoint from which metadata can be retrieved.

Location

A URI that specifies the location from which metadata can be retrieved.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_METADATA_SECTION_LIST structure (wsdypes.h)

Article02/22/2024

Represents a node in a single-linked list of metadata sections.

Syntax

C++

```
typedef struct _WSD_METADATA_SECTION_LIST {
    WSD_METADATA_SECTION_LIST *Next;
    WSD_METADATA_SECTION      *Element;
} WSD_METADATA_SECTION_LIST;
```

Members

Next

Reference to the next node in the linked list of **WSD_METADATA_SECTION_LIST** structures.

Element

The metadata section referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_NAME_LIST structure (wsdtypes.h)

Article02/22/2024

Represents a node in a single-linked list of XML name structures.

Syntax

C++

```
typedef struct _WSD_NAME_LIST {
    WSD_NAME_LIST *Next;
    WSDXML_NAME    *Element;
} WSD_NAME_LIST;
```

Members

Next

Reference to the next node in the linked list of **WSD_NAME_LIST** structures.

Element

The XML qualified name data referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?  

WSD_OPERATION structure (wsdtypes.h)

Article 02/22/2024

Describes an operation as defined by WSDL in terms of one or two messages. This structure is populated by [generated code](#).

Syntax

C++

```
typedef struct _WSD_OPERATION {  
    WSDXML_TYPE      *RequestType;  
    WSDXML_TYPE      *ResponseType;  
    WSD_STUB_FUNCTION RequestStubFunction;  
} WSD_OPERATION;
```

Members

RequestType

Reference to a [WSDXML_TYPE](#) structure that specifies the request type of an incoming message.

ResponseType

Reference to a [WSDXML_TYPE](#) structure that specifies the response type of an outgoing message.

RequestStubFunction

Reference to a [WSD_STUB_FUNCTION](#) function that specifies the address of a stub function which translates a generic SOAP message structure into a method call with a signature specific to the incoming message of the operation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_PORT_TYPE structure (wsdtypes.h)

Article02/22/2024

Supplies data about a port type. This structure is populated by [generated code](#).

Syntax

C++

```
typedef struct _WSD_PORT_TYPE {
    DWORD          EncodedName;
    DWORD          OperationCount;
    WSD_OPERATION  *Operations;
    WSD_PROTOCOL_TYPE ProtocolType;
} WSD_PORT_TYPE;
```

Members

EncodedName

The encoded qualified name of the port type.

OperationCount

The number of operations in the array referenced by the **Operations** member.

Operations

Reference to an array of [WSD_OPERATION](#) structures that specifies the operations comprising the port type.

ProtocolType

A [WSD_PROTOCOL_TYPE](#) value that specifies the protocol(s) supported by the port type.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_PROBE structure (wsdtypes.h)

Article02/22/2024

Represents a [Probe](#) message.

Syntax

C++

```
typedef struct _WSD_PROBE {
    WSD_NAME_LIST    *Types;
    WSD_SCOPES       *Scopes;
    WSDXML_ELEMENT   *Any;
} WSD_PROBE;
```

Members

Types

Reference to a [WSD_NAME_LIST](#) structure that contains a list of WS-Discovery Types.

Scopes

Reference to a [WSD_SCOPES](#) structure that contains a list of WS-Discovery Scopes.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Probe Message](#)

[ProbeMatches Message](#)

[WSD_PROBE_MATCH](#)

[WSD_PROBE_MATCHES](#)

[WSD_PROBE_MATCH_LIST](#)

Feedback

Was this page helpful?

 **Yes**

 **No**

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_PROBE_MATCH structure (wsdtypes.h)

Article 02/22/2024

Represents a ProbeMatch message.

Syntax

C++

```
typedef struct _WSD_PROBE_MATCH {  
    WSD_ENDPOINT_REFERENCE *EndpointReference;  
    WSD_NAME_LIST          *Types;  
    WSD_SCOPES             *Scopes;  
    WSD_URI_LIST           *XAddrs;  
    ULONGLONG              MetadataVersion;  
    WSDXML_ELEMENT         *Any;  
} WSD_PROBE_MATCH;
```

Members

EndpointReference

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies the matching endpoint.

Types

Reference to a [WSD_NAME_LIST](#) structure that contains a list of WS-Discovery Types.

Scopes

Reference to a [WSD_SCOPES](#) structure that contains a list of WS-Discovery Scopes.

XAddrs

Reference to a [WSD_URI_LIST](#) structure that contains a list of WS-Discovery XAddrs.

MetadataVersion

The metadata version of this message.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Probe Message](#)

[ProbeMatches Message](#)

[WSD_PROBE](#)

[WSD_PROBE_MATCHES](#)

[WSD_PROBE_MATCH_LIST](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_PROBE_MATCH_LIST structure (wsdtypes.h)

Article 10/05/2021

Represents a node in a single-linked list of ProbeMatch message structures.

Syntax

C++

```
typedef struct _WSD_PROBE_MATCH_LIST {  
    WSD_PROBE_MATCH_LIST *Next;  
    WSD_PROBE_MATCH *Element;  
} WSD_PROBE_MATCH_LIST;
```

Members

Next

Reference to the next node in the linked list of **WSD_PROBE_MATCH_LIST** structures.

Element

Reference to a [WSD_PROBE_MATCH](#) structure that contains the ProbeMatch message referenced by this node.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Probe Message](#)

ProbeMatches Message

WSD_PROBE

WSD_PROBE_MATCH

WSD_PROBE_MATCHES

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD_PROBE_MATCHES structure (wsdtypes.h)

Article 10/05/2021

Represents a [ProbeMatches](#) message.

Syntax

C++

```
typedef struct _WSD_PROBE_MATCHES {  
    WSD_PROBE_MATCH_LIST *ProbeMatch;  
    WSDXML_ELEMENT        *Any;  
} WSD_PROBE_MATCHES;
```

Members

ProbeMatch

Reference to a [WSD_PROBE_MATCH_LIST](#) structure that contains the list of matches to the [Probe](#) message.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Probe Message](#)

ProbeMatches Message

WSD_PROBE

WSD_PROBE_MATCH

WSD_PROBE_MATCH_LIST

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD_REFERENCE_PARAMETERS structure (wsdtypes.h)

Article02/22/2024

Specifies opaque data that is used by an endpoint.

Syntax

C++

```
typedef struct _WSD_REFERENCE_PARAMETERS {
    WSDXML_ELEMENT *Any;
} WSD_REFERENCE_PARAMETERS;
```

Members

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[WSD_ENDPOINT_REFERENCE](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_REFERENCE_PROPERTIES structure (wsdtypes.h)

Article02/22/2024

Specifies additional data used to uniquely identify an endpoint.

Syntax

C++

```
typedef struct _WSD_REFERENCE_PROPERTIES {
    WSDXML_ELEMENT *Any;
} WSD_REFERENCE_PROPERTIES;
```

Members

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[WSD_ENDPOINT_REFERENCE](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) 

| [Get help at Microsoft Q&A](#)

WSD_RELATIONSHIP_METADATA structure (wsdtypes.h)

Article02/22/2024

Provides metadata about the relationship between two or more services.

Syntax

C++

```
typedef struct _WSD_RELATIONSHIP_METADATA {
    const WCHAR          *Type;
    WSD_HOST_METADATA    *Data;
    WSDXML_ELEMENT       *Any;
} WSD_RELATIONSHIP_METADATA;
```

Members

Type

A WS-Discovery Type.

Data

Reference to a [WSD_HOST_METADATA](#) structure that contains metadata for all services hosted by a device.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_RESOLVE structure (wsdtypes.h)

Article02/22/2024

Represents a [Resolve](#) message.

Syntax

C++

```
typedef struct _WSD_RESOLVE {
    WSD_ENDPOINT_REFERENCE *EndpointReference;
    WSDXML_ELEMENT         *Any;
} WSD_RESOLVE;
```

Members

EndpointReference

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies the endpoint to match.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[Resolve Message](#)

WSD_RESOLVE_MATCH

WSD_RESOLVE_MATCHES

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_RESOLVE_MATCH structure (wsdtypes.h)

Article 02/22/2024

Represents a ResolveMatch message.

Syntax

C++

```
typedef struct _WSD_RESOLVE_MATCH {  
    WSD_ENDPOINT_REFERENCE *EndpointReference;  
    WSD_NAME_LIST           *Types;  
    WSD_SCOPES              *Scopes;  
    WSD_URI_LIST            *XAddrs;  
    ULONGLONG               MetadataVersion;  
    WSDXML_ELEMENT          *Any;  
} WSD_RESOLVE_MATCH;
```

Members

EndpointReference

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies the matching endpoint.

Types

Reference to a [WSD_NAME_LIST](#) structure that contains a list of WS-Discovery Types.

Scopes

Reference to a [WSD_SCOPES](#) structure that contains a list of WS-Discovery Scopes.

XAddrs

Reference to a [WSD_URI_LIST](#) structure that contains a list of WS-Discovery XAddrs.

MetadataVersion

The metadata version of this message.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

See also

[ResolveMatches Message](#)

[WSD_RESOLVE](#)

[WSD_RESOLVE_MATCHES](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_RESOLVE_MATCHES structure (wsdtypes.h)

Article02/22/2024

Represents a [ResolveMatches](#) message.

Syntax

C++

```
typedef struct _WSD_RESOLVE_MATCHES {
    WSD_RESOLVE_MATCH *ResolveMatch;
    WSDXML_ELEMENT     *Any;
} WSD_RESOLVE_MATCHES;
```

Members

ResolveMatch

Reference to a [WSD_RESOLVE_MATCH](#) structure that contains a child ResolveMatch message.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

ResolveMatches Message

WSD_RESOLVE

WSD_RESOLVE_MATCH

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SCOPES structure (wsdtypes.h)

Article02/22/2024

A collection of scopes used in WS-Discovery messaging.

Syntax

C++

```
typedef struct _WSD_SCOPES {
    const WCHAR *MatchBy;
    WSD_URI_LIST *Scopes;
} WSD_SCOPES;
```

Members

MatchBy

A matching rule used for scopes.

Scopes

Reference to a [WSD_URI_LIST](#) structure that contains a list of scopes.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

WSD_SECURITY_CERT_VALIDATION structure (wsdbase.h)

Article 07/27/2022

Represents the criteria for matching client certificates against those of an HTTPS server.

Do not use [WSD_SECURITY_CERT_VALIDATION_V1](#) directly in your code; using **WSD_SECURITY_CERT_VALIDATION** instead ensures that the proper version, based on the Windows version.

Syntax

C++

```
typedef struct _WSD_SECURITY_CERT_VALIDATION {
    PCCERT_CONTEXT *certMatchArray;
    DWORD          dwCertMatchArrayCount;
    HCERTSTORE      hCertMatchStore;
    HCERTSTORE      hCertIssuerStore;
    DWORD           dwCertCheckOptions;
    LPCWSTR         pszCNGHashAlgId;
    BYTE            *pbCertHash;
    DWORD           dwCertHashSize;
} WSD_SECURITY_CERT_VALIDATION;
```

Members

certMatchArray

An array of [CERT_CONTEXT](#) structures that contain certificates to be matched against those provided by the HTTPS server or client. Only one matching certificate is required for validation. This parameter can be NULL.

dwCertMatchArrayCount

The count of certificates in *certMatchArray*.

hCertMatchStore

A handle to a certificate store that contains certificates to be matched against those provided by the HTTPS server or client. Only one matching certificate is required for validation. This parameter can be NULL.

hCertIssuerStore

A handle to a certificate store that contains root certificates against which a certificate from the HTTPS server or client should chain to. Validation succeeds as long as the certificate chains up to at least one root certificate. This parameter can be NULL.

dwCertCheckOptions

A bitwise OR combination of values that specify which certificate checks to ignore.

Value	Meaning
WSDAPI_SSL_CERT_DEFAULT_CHECKS 0x0	Handle any revoked certificate errors.
WSDAPI_SSL_CERT_IGNORE_REVOCATION 0x1	Ignore revoked certificate errors.
WSDAPI_SSL_CERT_IGNORE_EXPIRY 0x2	Ignore expired certificate errors.
WSDAPI_SSL_CERT_IGNORE_WRONG_USAGE 0x4	Ignore certificate use errors.
WSDAPI_SSL_CERT_IGNORE_UNKNOWN_CA 0x8	Ignore unknown certificate authority errors.
WSDAPI_SSL_CERT_IGNORE_INVALID_CN 0x10	Ignore invalid common name certificate errors.

pszCNGHashAlgId

pbCertHash

dwCertHashSize

Remarks

This structure is used in the **pConfigData** member of the [WSD_CONFIG_PARAM](#) structure.

When the **configParamType** of [WSD_CONFIG_PARAM](#) is **WSD_SECURITY_SSL_SERVER_CERT_VALIDATION**, this structure can be used to validate SSL server certificates presented by an SSL server.

When the **configParamType** of [WSD_CONFIG_PARAM](#) is **WSD_SECURITY_SSL_CLIENT_CERT_VALIDATION**, this structure can be used to validate

SSL client certificates presented by an SSL client.

WSD_SECURITY_CERT_VALIDATION defines 3 certificate matching mechanisms. To obtain a match, at least one such mechanism must be satisfied.

If the application is built using Windows 8 SDK targeted for Windows 8 OS, **WSD_SECURITY_CERT_VALIDATION** resolves into the new structure. However, as a result, the application can then only run on Windows 8 machines.

If the application is built using Windows 8 SDK targeted for Windows 7 OS, **WSD_SECURITY_CERT_VALIDATION** will resolve into the old structure ([WSD_SECURITY_CERT_VALIDATION_V1](#)). While it's a given that the application will be supported for Windows 7, it also on Windows 8 since **wsdapi.dll** on Windows 8 will handle both the old and the newer versions of this structure.

An application already built using Windows 7 SDK will use the old version of this structure. It will run fine on Windows 8 since **wsdapi.dll** on Windows 8 can handle both versions.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	wsdbase.h (include Windows.h)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD_SECURITY_CERT_VALIDATION_V1 structure

Article 08/31/2015

Represents the criteria for matching client certificates against those of an HTTPS server.

Note

Do not implement this structure directly; implementing [WSD_SECURITY_CERT_VALIDATION](#) instead ensures that the proper structure version is used, based on the Windows platform the application code is compiled on.

Syntax

C++

```
typedef struct _WSD_SECURITY_CERT_VALIDATION_V1 {  
    struct WSD_SECURITY_CERT_VALIDATION;  
    LPCWSTR pszCNGHashAlgId;  
    BYTE     *pbCertHash;  
    DWORD    dwCertHashSize;  
} WSD_SECURITY_CERT_VALIDATION_V1, *PWSD_SECURITY_CERT_VALIDATION_V1;
```

Members

- [WSD_SECURITY_CERT_VALIDATION](#)
The default [WSD_SECURITY_CERT_VALIDATION](#) structure.
- **pszCNGHashAlgId**
The ID of the algorithm used to generate the client certificate hash.
- **pbCertHash**
Pointer to a buffer containing the client certificate hash. The buffer can be freed by the application as soon as [WSDCreateDeviceHost2](#) or [WSDCreateDeviceProxy2](#) return success; WSDAPI generates an internal copy of the buffer.
- **dwCertHashSize**
Indicates the length of *pbCertHash*.

Remarks

If the application is built using the Windows 8 SDK, targeted for Windows 8, [WSD_SECURITY_CERT_VALIDATION](#) resolves as **WSD_SECURITY_CERT_VALIDATION_V1**. The application can then only run on Windows 8 machines.

However, if the application is built using Windows 8 SDK, but targeted for Windows 7, [WSD_SECURITY_CERT_VALIDATION](#) will resolve to the old structure. While it's a given that the application will be supported for Windows 7, Windows 8 will also support the application as **wsdapi.dll** on Windows 8 will handle both the old and the newer versions of this structure. For example, an application developed with the Windows 7 SDK will run fine on Windows 8 machines.

Requirements

 Expand table

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	Wsdbase.h (include Windows.h)

See also

[WSD_SECURITY_CERT_VALIDATION](#)

[WSDCreateDeviceHost2](#)

[WSDCreateDeviceProxy2](#)

WSD_SECURITY_SIGNATURE_VALIDATION structure (wsdbase.h)

Article 02/22/2024

Represents the criteria for matching client compact signatures against messages.

Syntax

C++

```
typedef struct _WSD_SECURITY_SIGNATURE_VALIDATION {  
    PCCERT_CONTEXT *signingCertArray;  
    DWORD          dwSigningCertArrayCount;  
    HCERTSTORE      hSigningCertStore;  
    DWORD          dwFlags;  
} WSD_SECURITY_SIGNATURE_VALIDATION, *PWSD_SECURITY_SIGNATURE_VALIDATION;
```

Members

signingCertArray

An array of [CERT_CONTEXT](#) structures that contain certificates to be matched against a message. Only one matching certificate is required for validation. This parameter can be **NULL**.

dwSigningCertArrayCount

The count of certificates in *signingMatchArray*.

hSigningCertStore

A handle to a certificate store that contains certificates to be matched against a message. Only one matching certificate is required for validation. This parameter can be **NULL**.

dwFlags

A flag that specifies how unsigned messages are handled. If set to **WSDAPI_COMPACTSIG_ACCEPT_ALL_MESSAGES**, then the discovery object will accept unsigned messages, signed-and-verified messages and signed-but-verified, (that is, those for which the signing cert could not be found either in the store or the certificate

array) messages. If this flag is not set, then only the signed-and-verified messages will be accepted.

If `WSDAPI_COMPACTSIG_ACCEPT_ALL_MESSAGES` is specified, the caller will not be able use the [IWSDSignatureProperty](#) interface to learn whether the message was signed or not.

Remarks

This structure is used in the `pConfigData` member of the [WSD_CONFIG_PARAM](#) structure when `configParamType` is set to `WSD_SECURITY_COMPACTSIG_VALIDATION`.

`WSD_SECURITY_SIGNATURE_VALIDATION` defines 2 matching mechanisms. To obtain a match, at least one such mechanism must be satisfied.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	wsdbase.h (include Windows.h)

Feedback

Was this page helpful? Yes No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WSD_SERVICE_METADATA structure (wsdtypes.h)

Article 02/22/2024

Provides metadata regarding a service hosted by a device.

Syntax

C++

```
typedef struct _WSD_SERVICE_METADATA {  
    WSD_ENDPOINT_REFERENCE_LIST *EndpointReference;  
    WSD_NAME_LIST                *Types;  
    const WCHAR                  *ServiceId;  
    WSDXML_ELEMENT                *Any;  
} WSD_SERVICE_METADATA;
```

Members

EndpointReference

Reference to a [WSD_ENDPOINT_REFERENCE_LIST](#) structure that specifies the endpoints at which the service is available.

Types

Reference to a [WSD_NAME_LIST](#) structure that contains a list of WS-Discovery Types.

ServiceId

The URI of the service. This URI must be valid when a [WSD_SERVICE_METADATA](#) structure is passed to [IWSDDeviceHost::SetMetadata](#). Applications are responsible for URI validation.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SERVICE_METADATA_LIST structure (wsdtypes.h)

Article02/22/2024

Represents a node in a single-linked list of service metadata structures.

Syntax

C++

```
typedef struct _WSD_SERVICE_METADATA_LIST {
    WSD_SERVICE_METADATA_LIST *Next;
    WSD_SERVICE_METADATA      *Element;
} WSD_SERVICE_METADATA_LIST;
```

Members

Next

Reference to the next node in the linked list of **WSD_SERVICE_METADATA_LIST** structures.

Element

Reference to a [WSD_SERVICE_METADATA](#) structure that represents the service metadata referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SOAP_FAULT structure (wsdtypes.h)

Article 10/05/2021

Represents a generated SOAP fault.

Syntax

C++

```
typedef struct _WSD_SOAP_FAULT {  
    WSD_SOAP_FAULT_CODE    *Code;  
    WSD_SOAP_FAULT_REASON *Reason;  
    const WCHAR            *Node;  
    const WCHAR            *Role;  
    WSDXML_ELEMENT         *Detail;  
} WSD_SOAP_FAULT;
```

Members

Code

A [WSD_SOAP_FAULT_CODE](#) structure that contains a SOAP fault code.

Reason

A [WSD_SOAP_FAULT_REASON](#) structure that contains localized human readable explanations of the fault.

Node

The SOAP node on the SOAP message path that caused the fault.

Role

The SOAP role in which the **Node** was acting at the time the fault occurred.

Detail

A [WSDXML_ELEMENT](#) structure that contains application-specific error information pertaining to the fault.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[WSDGenerateFault](#)

[WSDGenerateFaultEx](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD_SOAP_FAULT_CODE structure (wsdtypes.h)

Article02/22/2024

Represents a generated SOAP fault code.

Syntax

C++

```
typedef struct _WSD_SOAP_FAULT_CODE {
    WSDXML_NAME          *Value;
    WSD_SOAP_FAULT_SUBCODE *Subcode;
} WSD_SOAP_FAULT_CODE;
```

Members

Value

A [WSDXML_NAME](#) structure that contains the qualified name of the SOAP fault code.

Subcode

A [WSD_SOAP_FAULT_SUBCODE](#) structure that contains the fault subcode.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[WSD_SOAP_FAULT](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SOAP_FAULT_REASON structure (wsdtypes.h)

Article02/22/2024

A collection of reason codes associated with a [WSD_SOAP_FAULT](#). A reason code is a human readable explanation of the fault.

Syntax

C++

```
typedef struct _WSD_SOAP_FAULT_REASON {
    WSD_LOCALIZED_STRING_LIST *Text;
} WSD_SOAP_FAULT_REASON;
```

Members

Text

A [WSD_LOCALIZED_STRING_LIST](#) structure that contains a collection of localized reason codes.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

[WSD_SOAP_FAULT](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SOAP_FAULT_SUBCODE structure (wsdtypes.h)

Article02/22/2024

Represents a generated SOAP fault subcode. Subcodes can be nested.

Syntax

C++

```
typedef struct _WSD_SOAP_FAULT_SUBCODE {
    WSDXML_NAME          *Value;
    WSD_SOAP_FAULT_SUBCODE *Subcode;
} WSD_SOAP_FAULT_SUBCODE;
```

Members

Value

A [WSDXML_NAME](#) structure that contains the qualified name of the SOAP fault subcode.

Subcode

A [WSD_SOAP_FAULT_SUBCODE](#) structure that contains a fault subcode.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

See also

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SOAP_HEADER structure (wsdtypes.h)

Article 02/22/2024

Provides SOAP header data for the [WSD_SOAP_MESSAGE](#) structure.

Syntax

C++

```
typedef struct _WSD_SOAP_HEADER {  
    const WCHAR          *To;  
    const WCHAR          *Action;  
    const WCHAR          *MessageID;  
    WSD_HEADER_RELATESTO RelatesTo;  
    WSD_ENDPOINT_REFERENCE *ReplyTo;  
    WSD_ENDPOINT_REFERENCE *From;  
    WSD_ENDPOINT_REFERENCE *FaultTo;  
    WSD_APP_SEQUENCE      *AppSequence;  
    WSDXML_ELEMENT        *AnyHeaders;  
} WSD_SOAP_HEADER;
```

Members

To

The URI to which the SOAP message is addressed.

Action

The action encoded by the SOAP message.

MessageID

An identifier that distinguishes the message from others from the same sender.

RelatesTo

In response messages, specifies the message ID of the matching request message.

ReplyTo

In request messages, a reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies to the endpoint to which responses should be sent.

From

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies the endpoint from which the SOAP message was sent.

FaultTo

Reference to a [WSD_ENDPOINT_REFERENCE](#) structure that specifies to the endpoint to which fault messages should be sent.

AppSequence

In discovery messages, a reference to a [WSD_APP_SEQUENCE](#) structure that helps the recipient determine the order in which messages were issued by the sender.

AnyHeaders

Reference to a [WSDXML_ELEMENT](#) structure that specifies additional headers not encoded by the other members.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SOAP_MESSAGE structure (wsdtypes.h)

Article02/22/2024

The contents of a WSD SOAP message. This structure is used for [Probe](#) messages, [ProbeMatch](#) messages, [Resolve](#) messages, and [ResolveMatch](#) messages, among others.

Syntax

C++

```
typedef struct _WSD_SOAP_MESSAGE {
    WSD_SOAP_HEADER Header;
    void *Body;
    WSDXML_TYPE *BodyType;
} WSD_SOAP_MESSAGE;
```

Members

Header

A [WSD_SOAP_HEADER](#) structure that specifies the header of the SOAP message.

Body

The body of the SOAP message.

BodyType

Reference to a [WSDXML_TYPE](#) structure that specifies the type of the SOAP message body.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Header	wsdtypes.h (include Wsdapi.h)

See also

[Discovery and Metadata Exchange Messages](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_SYNCHRONOUS_RESPONSE_CONTEXT structure (wsdtypes.h)

Article 02/22/2024

Provides a context for handling the response to a two-way request.

Syntax

C++

```
typedef struct _WSD_SYNCHRONOUS_RESPONSE_CONTEXT {  
    HRESULT          hr;  
    HANDLE           eventHandle;  
    struct IWSDMessageParameters *messageParameters;  
    void             *results;  
} WSD_SYNCHRONOUS_RESPONSE_CONTEXT;
```

Members

hr

The result code of the last operation performed using this response context.

eventHandle

The event handle to be signaled when the response is ready.

messageParameters

A pointer to an [IWSDMessageParameters](#) object that contains transport information associated with the response.

results

The body of the response message.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_THIS_DEVICE_METADATA structure (wsdtypes.h)

Article 10/06/2021

Specifies metadata that is unique to a specific device.

Syntax

C++

```
typedef struct _WSD_THIS_DEVICE_METADATA {  
    WSD_LOCALIZED_STRING_LIST *FriendlyName;  
    const WCHAR                *FirmwareVersion;  
    const WCHAR                *SerialNumber;  
    WSDXML_ELEMENT             *Any;  
} WSD_THIS_DEVICE_METADATA;
```

Members

FriendlyName

Reference to a [WSD_LOCALIZED_STRING_LIST](#) structure that contains the list of localized friendly names for the device. It should be set to fewer than 256 characters.

FirmwareVersion

The firmware version of the device. It should be set to fewer than 256 characters.

SerialNumber

The serial number of the device. It should be set to fewer than 256 characters.

Any

Reference to a [WSDXML_ELEMENT](#) structure that provides an extensible space for devices to add custom metadata to the device specific section. For example, you can use this to add a user-defined name for the device.

Remarks

ThisDevice metadata follows this form:

syntax

```
<wsd:ThisDevice>
  <wsd:FriendlyName>
    A. Datum WebWeigh Scale
  </wsd:FriendlyName>
  <wsd:FirmwareVersion>
    2.53c
  </wsd:FirmwareVersion>
  <wsd:SerialNumber>
    923450982349058
  </wsd:SerialNumber>
</wsd:ThisDevice>
```

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSD_THIS_MODEL_METADATA structure (wsdtypes.h)

Article 02/22/2024

Provides model-specific information relating to the device.

Syntax

C++

```
typedef struct _WSD_THIS_MODEL_METADATA {  
    WSD_LOCALIZED_STRING_LIST *Manufacturer;  
    const WCHAR                *ManufacturerUrl;  
    WSD_LOCALIZED_STRING_LIST *ModelName;  
    const WCHAR                *ModelNumber;  
    const WCHAR                *ModelUrl;  
    const WCHAR                *PresentationUrl;  
    WSDXML_ELEMENT             *Any;  
} WSD_THIS_MODEL_METADATA;
```

Members

Manufacturer

Reference to a [WSD_LOCALIZED_STRING_LIST](#) structure that contains the manufacturer name. The name should be set to fewer than 2048 characters.

ManufacturerUrl

The URL to a Web site for the device manufacturer. The URL should have fewer than 2048 characters.

ModelName

Reference to a [WSD_LOCALIZED_STRING_LIST](#) structure that specifies model names. This is a list of localized friendly names that should be set to fewer than 256 characters.

ModelNumber

The model number. This should be set to fewer than 256 characters.

ModelUrl

The URL to a Web site for this device model. The URL should have fewer than 2048 characters.

PresentationUrl

An HTML page for this device. This can be relative to a base URL set by XML Base. The URL should have fewer than 2048 characters.

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML **ANY** keyword.

Remarks

WSD_THIS_MODEL_METADATA specifies manufacturer metadata that is common to all instances of a specific model.

Model metadata follows this form:

syntax

```
<wsd:ThisModel>
  <wsd:Manufacturer>
    A. Datum Corporation
  </wsd:Manufacturer>
  <wsd:ManufacturerURL>
    http://www.adatum.com
  </wsd:ManufacturerURL>
  <wsd:ModelName>
    WebWeigh
  </wsd:ModelName>
  <wsd:ModelNumber>
    9-23492-83049
  </wsd:ModelNumber>
  <wsd:ModelURL>
    http://www.adatum.com/WebWeighOwner.html
  </wsd:ModelURL>
  <wsd:PresentationURL>
    presentation/menu.html
  </wsd:PresentationURL>
</wsd:ThisModel>
```

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSD_UNKNOWN_LOOKUP structure (wsdtypes.h)

Article02/22/2024

Represents an XML element that could not be parsed.

Syntax

C++

```
typedef struct _WSD_UNKNOWN_LOOKUP {
    WSDXML_ELEMENT *Any;
} WSD_UNKNOWN_LOOKUP;
```

Members

Any

Reference to a [WSDXML_ELEMENT](#) structure that specifies extension content allowed by the XML ANY keyword.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdtypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

WSD_URI_LIST structure (wsdypes.h)

Article02/22/2024

Represents a node in a linked list of URIs.

Syntax

C++

```
typedef struct _WSD_URI_LIST {
    WSD_URI_LIST *Next;
    const WCHAR *Element;
} WSD_URI_LIST;
```

Members

Next

Reference to the next node in the single-linked list of **WSD_URI_LIST** structures.

Element

The URI referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdypes.h (include Wsdapi.h)

Feedback

Was this page helpful?

 Yes

 No

WSDUdpRetransmitParams structure (wsdbase.h)

Article 02/22/2024

Defines the parameters for repeating a message transmission.

Syntax

C++

```
typedef struct _WSDUdpRetransmitParams {  
    ULONG ulSendDelay;  
    ULONG ulRepeat;  
    ULONG ulRepeatMinDelay;  
    ULONG ulRepeatMaxDelay;  
    ULONG ulRepeatUpperDelay;  
} WSDUdpRetransmitParams;
```

Members

ulSendDelay

Time to wait before sending the first transmission, in milliseconds. Specify zero for no delay. Cannot be INFINITE.

ulRepeat

Maximum number of transmissions to send. Specify a value between 1 and 256, inclusively.

ulRepeatMinDelay

Minimum value of the range used to generate the initial delay value, in milliseconds. This value must be less than or equal to **ulRepeatMaxDelay**, can be zero, but cannot be INFINITE. See Remarks.

ulRepeatMaxDelay

Maximum value of the range used to generate the initial delay value, in milliseconds. This value be less than or equal to **ulRepeatUpperDelay**, can be zero, but cannot be INFINITE. See Remarks.

Maximum delay to wait before sending message, in milliseconds. This value can be zero, but cannot be INFINITE.

Remarks

If `ulRepeatMinDelay`, `ulRepeatMaxDelay`, and `ulRepeatUpperDelay` are all zero, there is no delay in retransmission of the message.

WSD sends the first transmission after waiting `ulSendDelay`. WSD uses the other members to determine when to repeat the transmission, if necessary. WSD repeats the transmission up to `ulRepeat` times with increasing delays between transmission. WSD uses the `ulRepeatMinDelay`, `ulRepeatMaxDelay`, and `ulRepeatUpperDelay` members to determine the delay.

WSD generates a random delay value in the range `ulRepeatMinDelay` to `ulRepeatMaxDelay` and waits this amount of time before repeating the transmission. All subsequent repeat attempts then double the current delay value until `ulRepeatUpperDelay` is reached. For example, if the initial random delay value is 50 and the upper delay value is 250, the second attempt will wait 50 milliseconds, the third attempt will wait 100 milliseconds, the fourth attempt will wait 200 milliseconds, and the remaining attempts will wait 250 milliseconds.

For details on how WSD uses these values to send messages, see Appendix I of the [SOAP-over-UDP](#) specification.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdbase.h (include Wsdapi.h)

See also

[IWSDUdpMessageParameters::GetRetransmitParams](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDXML_ATTRIBUTE structure (wsdxmldom.h)

Article 02/22/2024

Describes an XML attribute.

Syntax

C++

```
typedef struct _WSDXML_ATTRIBUTE {  
    WSDXML_ELEMENT *Element;  
    WSDXML_ATTRIBUTE *Next;  
    WSDXML_NAME *Name;  
    WCHAR *Value;  
} WSDXML_ATTRIBUTE;
```

Members

Element

Reference to a [WSDXML_ELEMENT](#) structure that specifies parent element of the attribute.

Next

Reference to a [WSDXML_ATTRIBUTE](#) structure that specifies the next sibling attribute, if any.

Name

Reference to a [WSDXML_NAME](#) structure that specifies the qualified name of the attribute.

Value

The value of the attribute.

Remarks

[WSDXML_ATTRIBUTE](#) is used to describe attribute values in an XML element.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxml.dom.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDXML_ELEMENT_LIST structure (wsdxml.h)

Article02/22/2024

Represents a node in a linked list of XML elements.

Syntax

C++

```
typedef struct _WSDXML_ELEMENT_LIST {
    WSDXML_ELEMENT_LIST *Next;
    WSDXML_ELEMENT      *Element;
} WSDXML_ELEMENT_LIST;
```

Members

Next

Reference to the next node in the linked list of **WSDXML_ELEMENT_LIST** structures.

Element

The **WSDXML_ELEMENT** structure referenced by this node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxml.h

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) 

| [Get help at Microsoft Q&A](#)

WSDXML_ELEMENT structure (wsdxml.h)

Article 10/05/2021

Describes an XML element.

Syntax

C++

```
typedef struct _WSDXML_ELEMENT {  
    WSDXML_NODE        Node;  
    WSDXML_NAME        *Name;  
    WSDXML_ATTRIBUTE    *FirstAttribute;  
    WSDXML_NODE        *FirstChild;  
    WSDXML_PREFIX_MAPPING *PrefixMappings;  
} WSDXML_ELEMENT;
```

Members

Node

Reference to a [WSDXML_NODE](#) structure that specifies the parent element, next sibling and type of the node.

Name

Reference to a [WSDXML_NAME](#) structure that specifies name.

FirstAttribute

Reference to a [WSDXML_ATTRIBUTE](#) structure that specifies the first attribute.

FirstChild

Reference to a [WSDXML_NODE](#) structure that specifies the first child.

PrefixMappings

Reference to a [WSDXML_PREFIX_MAPPING](#) structure that specifies the prefix mappings.

Remarks

WSDXML_ELEMENT represents an XML element in the DOM tree. The **Name** member can be used to determine the name and namespace of this element. **FirstAttribute** points to any attributes, and **FirstChild** points to anything contained within the element.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxmldom.h

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSDXML_NAME structure (wsdxml.h)

Article 02/22/2024

Specifies an XML qualified name.

Syntax

C++

```
typedef struct _WSDXML_NAME {  
    WSDXML_NAMESPACE *Space;  
    WCHAR             *LocalName;  
} WSDXML_NAME;
```

Members

Space

Reference to a [WSDXML_NAMESPACE](#) structure that specifies the namespace of the qualified name.

LocalName

The local name of the qualified name.

Remarks

WSDXML_NAME represents a single name within a namespace. The relationship between the name and namespace is circular, in that the **Space** pointer of the name points to the namespace the name belongs to, and the **Names** array of the namespace will have an entry for the name.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxmlDOM.h

Feedback

Was this page helpful?



[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDXML_NAMESPACE structure (wsdxml.h)

Article 10/05/2021

Specifies an XML namespace.

Syntax

C++

```
typedef struct _WSDXML_NAMESPACE {  
    const WCHAR *Uri;  
    const WCHAR *PreferredPrefix;  
    WSDXML_NAME *Names;  
    WORD        NamesCount;  
    WORD        Encoding;  
} WSDXML_NAMESPACE;
```

Members

Uri

The URI that identifies the namespace.

PreferredPrefix

The preferred prefix to be used in XML prefix mappings.

Names

Reference to an array of [WSDXML_NAME](#) structures that specify the names in the namespace.

NamesCount

The number of names in the **Names** array.

Encoding

The encoded reference for the namespace.

Remarks

WSDXML_NAMESPACE represents the association between a namespace URI and a list of names belonging to that namespace. Additionally, it provides a **PreferredPrefix** for the namespace, which gives guidance on the default prefix to use for a specified namespace. In the context of WSDAPI, there are two types of namespaces: static namespaces and dynamic namespaces.

Static namespaces are user provided, well known, and assumed to be complete namespaces, in that all names belonging to the namespace should be in names array. When processing a received XML document, any element or attribute in the document that claims to be in a static namespace but has a name not listed in that namespace is treated as an error. Static namespaces are typically generated pre-compile time, by a tool like WSDCodeGen.

Dynamic namespaces are generated by WSDAPI. These are built when new namespaces are seen in XML documents. With dynamic namespaces, no assumptions can be made about whether a specified name actually belongs to the formal namespace or not, so all names are accepted as being part of the namespace. As such, dynamic namespaces expand the **Names** array as they process new names in a specified document.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxmlDom.h

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSDXML_NODE structure (wsdxml.h)

Article 02/22/2024

Describes an XML node.

Syntax

C++

```
typedef struct _WSDXML_NODE {  
    WSDXML_ELEMENT *Parent;  
    WSDXML_NODE     *Next;  
} WSDXML_NODE;
```

Members

Parent

Reference to the parent node in a linked list of [WSDXML_ELEMENT](#) structures.

Next

Reference to the next node in the linked list of **WSDXML_NODE** structures.

Remarks

WSDXML_NODE represents an arbitrary node within the DOM tree. Nodes are weakly typed; the **Type** member must be inspected to determine the actual type of the node, and the node pointer must then be cast to the structure of the appropriate type (see [WSDXML_ELEMENT](#) and [WSDXML_TEXT](#)) to obtain the node contents. **Parent** points to the containing element for the current node, and **Next** points to any nodes at the same level as the current node.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxmlDOM.h

Feedback

Was this page helpful?



[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDXML_PREFIX_MAPPING structure (wsdxml.h)

Article 02/22/2024

Describes an XML namespace prefix.

Syntax

C++

```
typedef struct _WSDXML_PREFIX_MAPPING {  
    DWORD                Refs;  
    WSDXML_PREFIX_MAPPING *Next;  
    WSDXML_NAMESPACE     *Space;  
    WCHAR                *Prefix;  
} WSDXML_PREFIX_MAPPING;
```

Members

Refs

The number of references to the mapping. When the value reaches zero, the mapping is deleted.

Next

Reference to the next node in a linked list of **WSDXML_PREFIX_MAPPING** structures.

Space

Reference to a [WSDXML_NAMESPACE](#) structure.

Prefix

The text of the XML prefix.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxmlDOM.h

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDXML_TEXT structure (wsdxml.h)

Article 02/22/2024

Describes the text in an XML node.

Syntax

C++

```
typedef struct _WSDXML_TEXT {  
    WSDXML_NODE Node;  
    WCHAR      *Text;  
} WSDXML_TEXT;
```

Members

Node

The current node in a linked list of [WSDXML_NODE](#) structures.

Text

The text contained in the XML node. The maximum length of this string is WSD_MAX_TEXT_LENGTH (8192). The text must consist of UTF-16 encoded characters. The text cannot contain raw XML, as special characters are rendered using the equivalent entity reference. For example, `<` is rendered as `<`.

Remarks

WSDXML_TEXT is used to represent the contents of an element. Since no type information exists for elements in DOM, the **Text** member is the exact representation of the element contents from the XML document.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxmlDOM.h

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WSDXML_TYPE structure (wsdxml.h)

Article02/22/2024

Describes an XSD type. This structure is populated by [generated code](#).

Syntax

C++

```
typedef struct _WSDXML_TYPE {
    const WCHAR *Uri;
    const BYTE *Table;
} WSDXML_TYPE;
```

Members

Uri

The optional URI that identifies the type.

Table

The type table that describes the schema of the type and its binary representation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wsdxml.h

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

WsdCodeGen Reference

Article • 01/07/2021

- [WsdCodeGen Command Line Syntax](#)
- [WsdCodeGen Configuration File XML Reference](#)
- [WSDAPI XML Type Bytecodes](#)

Related topics

[Web Services on Devices Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WsdCodeGen Command Line Syntax

Article • 01/07/2021

WsdCodeGen has two functions: generating configuration files and generating source code for WSDAPI client and host applications. This topic gives the command line syntax used to perform each task.

Generating a Configuration File

Syntax

WSDCODEGEN.EXE /generateconfig:{client|host|all} [/outputfile:*ConfigFileName*]
WSDLFileNameList

Parameters

/generateconfig:{client | host | all}

The type of code that the output configuration file will generate. **/generateconfig:client** is used to generate a configuration file for generating client code, **/generateconfig:host** is used to generate a configuration file for generating host code, and **/generateconfig:all** is used to generate a configuration file for generating both client and host code.

****/outputfile:***ConfigFileName***

This optional parameter is used to specify the file name of the output configuration file. If this parameter is excluded, the name of the output configuration file is `codegen.config`.

/pnpx

Include a PnP-X template in the configuration file.

WSDLFileNameList

A space-delimited list of the WSDL file(s) to be processed by WsdCodeGen.

Generating Source Code

Syntax

WSDCODEGEN.EXE [/generatecode [/download] [/gbc] [outputroot:*path*]
[/writeaccess:*command*] *ConfigFileName*

Parameters

/generatecode

Directs WsdCodeGen to generate source code. This is the default mode if no mode is specified.

/download

Downloads imported documents referenced by the configuration file. This parameter is optional.

/gbc

Adds comments to the source code that indicates the code was generated. These comments are prefixed with the phrase "Generated by". This parameter is optional.

****/outputroot:***path***

The output location for generated files. *path* can be an absolute or relative path. This parameter is optional.

****/writeaccess:***command***

Directs WsdCodeGen to execute the specified command before it modifies any existing files on disk. Output files that are identical to those on disk will not receive this command, nor will they be written to. If the command contains the sequence "{0}", this sequence will be replaced with the filename of the file to be modified. If not, the filename will be appended to the command.

Examples:

/writeaccess:"attrib -r"

/writeaccess:"attrib -r {0}"

/writeaccess:"copy {0} ..\backup\"

ConfigFileName

The name of the configuration file to process before generating code.

Formatting Legend

Format	Meaning
<i>Italic</i>	Information that the user must provide
Bold	Elements that the user must type exactly as shown
Between brackets ([])	Optional items
Between braces ({}); choices separated by pipe (). Example: {even odd}	Set of choices from which the user must choose only one

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

See also

[WsdCodeGen Configuration File](#)

[Using WsdCodeGen](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WsdCodeGen Configuration File XML Reference

Article • 01/07/2021

WsdCodeGen configuration files use the following XML elements.

- [async](#)
- [autoStatic](#)
- [codeName](#)
- [deallocators](#)
- [enumerationValueDeclarations](#)
- [eventArg](#)
- [events](#)
- [eventSourceBuilderDeclarations](#)
- [eventSourceBuilderImplementations](#)
- [eventStubFunction](#)
- [excludeImport](#)
- [faultInfo](#)
- [file](#)
- [functionDeclarations](#)
- [host](#)
- [hostBuilderDeclaration](#)
- [hostBuilderImplementation](#)
- [hosted](#)
- [hostMetadata](#)
- [hostedService](#)
- [idlFunctionDeclarations](#)
- [importHint](#)
- [include](#)
- [interface](#)
- [IUnknownDeclarations](#)
- [IUnknownDefinitions](#)
- [layerNumber](#)
- [layerPrefix](#)
- [literalInclude](#)
- [location](#)
- [macro](#)
- [macroPrefix](#)
- [manufacturer](#)

- manufacturerLS
- manufacturerURL
- messageStructureDefinitions
- messageTypeDeclarations
- messageTypeDefinitions
- modelName
- modelNameLS
- modelNumber
- modelURL
- name
- nameSpace
- namespaceDeclarations
- namespaceDefinitions
- notificationInterface
- operation
- operationDeallocator
- path
- PnPCompatibleId
- PnPDeviceCategory
- PnPHardwareId
- portType
- portTypeDeclarations
- portTypeDefinitions
- preferredPrefix
- presentationURL
- proxyBuilderDeclarations
- proxyBuilderImplementations
- proxyClass
- proxyFunctionImplementations
- relationshipMetadata
- relationshipMetadataDeclaration
- relationshipMetadataDefinition
- serverClass
- ServiceID
- structDeclarations
- structDefinitions
- stubDeclarations
- stubDefinitions
- subscriptionFunctionDeclarations
- subscriptionIdlFunctionDeclarations

- [subscriptionProxyFunctionImplementations](#)
- [thisModelMetadata](#)
- [thisModelMetadataDeclaration](#)
- [thisModelMetadataDefinition](#)
- [Types](#)
- [typeTableDeclarations](#)
- [typeTableDefinitions](#)
- [typeUri](#)
- [wsdCodeGen](#)
- [wsdl](#)
- [xsd](#)

Related topics

[WsdCodeGen Reference](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

async element

Article • 04/26/2021

Specifies whether asynchronous operations are included in the generated proxy functions.

Usage

syntax

```
<async/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
functionDeclarations	Generates implementation declarations for proxy functions for port type operations.
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.

Remarks

Possible values are 1 (asynchronous operations included) and 0 (default, asynchronous operations excluded).

A proxy can have both asynchronous and synchronous versions of the same operations.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

autoStatic element

Article • 04/26/2021

Indicates whether or not WsdCodeGen should try to automatically flag certain generated fields as static. This behavior is enabled by default.

Usage

syntax

```
<autoStatic/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

The **autoStatic** element is not required and may be omitted from the XML configuration file. The element can be used to disable the flagging of generated fields as static or to explicitly force the flagging of certain generated fields as static.

Possible values are 1 (enabled, default) and 0 (disabled). Enabling **autoStatic** may cause compilation issues depending on how the code generator is directed to output data.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

codeName element

Article • 04/26/2021

Defines the name to be used to identify the namespace in generated code.

Usage

syntax

```
<codeName/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
nameSpace	A namespace to be used for code generation.
portTypeDeclarations	Generates C constant declarations for port types.
portTypeDefinitions	Generates C constants for port types.

Remarks

This element overrides the default code name used for generated code. By default, the code generated creates a code name from the URI or qualified name.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

deallocator element

Article • 08/25/2021

Specifies the type of deallocator to be used by a stub function.

Usage

syntax

```
<deallocator/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
stubDefinitions	Generates implementations for stub functions for port-type operations.

Remarks

The deallocator type should be enclosed in a pair of <deallocator> tags. The following strings are valid deallocator values:

- none
- WSDFreeLinkedMemory
- CoTaskMemFree
- free
- delete
- deleteArray
- Release

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

enumerationValueDeclarations element

Article • 04/26/2021

Generates C declarations for values of all enumerated types.

Usage

syntax

```
<enumerationValueDeclarations/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 yes

 no

[Get help at Microsoft Q&A](#)

eventArg element

Article • 04/26/2021

Specifies whether related event arguments are included in the generated functions.

Usage

syntax

```
<eventArg/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
stubDefinitions	Generates implementations for stub functions for port type operations.

Remarks

Possible values are 1 (event arguments included) and 0 (default, event arguments excluded).

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

events element

Article • 04/26/2021

Specifies whether related events are included in the generated functions.

Usage

syntax

```
<events/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
functionDeclarations	Generates implementation declarations for proxy functions for port type operations.
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.
stubDeclarations	Generates declarations for stub functions for port type operations.
stubDefinitions	Generates implementations for stub functions for port type operations.

Remarks

Possible values are 1 (events included) and 0 (default, events excluded).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

eventSourceBuilderDeclarations element

Article • 04/26/2021

Generates declarations for functions that create event source classes.

Usage

syntax

```
<eventSourceBuilderDeclarations>  
  child elements  
</eventSourceBuilderDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
proxyClass	Class name to generate from the event source builder function.

Child element sequence

syntax

proxyClass

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

eventSourceBuilderImplementations element

Article • 04/26/2021

Generates functions that create event source classes.

Usage

syntax

```
<eventSourceBuilderImplementations>  
  child elements  
</eventSourceBuilderImplementations>
```

Attributes

There are no attributes.

Child elements

Element	Description
proxyClass	Class name to generate from the event source builder function.

Child element sequence

syntax

proxyClass

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

eventStubFunction element

Article • 04/26/2021

Specifies whether stub function references should be included in the operation structures in the port type definitions for notification operations.

Usage

syntax

```
<eventStubFunction/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
portTypeDefinitions	Generates C constants for port types.

Remarks

Stub function references are in client scenarios for notification operations (events).

Valid values for this element are 1 (TRUE/stub function references included) and 0 (FALSE/no stub function references included).

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

excludeImport element

Article • 04/26/2021

Prevents the generation of import statements for specified targets named in a `<wsdl:import>` element in a WSDL file. This can be used to keep WsdCodeGen from importing well-known targets, such as the WS-Addressing and WS-Eventing specifications, even though these targets are referenced in the WSDL.

The value of this element must be set to the namespace named in the `<wsdl:import>` element to exclude.

Usage

syntax

```
<excludeImport/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
wsdl	Specifies a WSDL file to process for contract information.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

faultInfo element

Article • 04/26/2021

Specifies whether parameters used to pass fault information are included in generated functions.

Usage

syntax

```
<faultInfo/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
functionDeclarations	Generates implementation declarations for proxy functions for port type operations.
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.
stubDefinitions	Generates implementations for stub functions for port type operations.

Remarks

Possible values are 1 (fault parameters included) and 0 (fault parameters excluded). By default, fault parameters are excluded.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

file element

Article • 04/26/2021

Directs the code generator to generate a file and specifies the output file name.

Usage

syntax

```
<file
  name = "pathname string">
  child elements
</file>
```

Attributes

Attribute	Type	Required	Description
name	pathname string	Yes	The output filename for the generated content. The filename string should include complete path information.

Child elements

Element	Description
CDATA	Text and CDATA sections are copied to the file without modification. Source code that is not a function of the contract input data can be added to output files using text and CDATA sections.
enumerationValueDeclarations	Generates C declarations for values of all enumerated types.
eventSourceBuilderDeclarations	Generates declarations for functions that create event source classes.
eventSourceBuilderImplementations	Generates functions that create event source classes.

Element	Description
functionDeclarations	Generates implementation declarations for proxy functions for port type operations.
hostBuilderDeclaration	Generates a declaration for a function that creates a typed host.
hostBuilderImplementation	Generates a function that creates a typed host.
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
include	Includes the contents of a macro or file in the generated output.
IUnknownDeclarations	Generates declarations for QueryInterface, AddRef and Release.
IUnknownDefinitions	Generates implementations for QueryInterface, AddRef and Release.
literalInclude	Places a C or IDL include statement in the generated code.
messageStructureDefinitions	Generates C structure definitions for message types.
messageTypeDeclarations	Generates C constant declarations for XML schema tables for message types.
messageTypeDefinitions	Generates C constants for XML schema tables for message types.
namespaceDeclarations	Generates C declarations for namespace tables.
namespaceDefinitions	Generates C definitions for namespace tables.

Element	Description
portTypeDeclarations	Generates C constant declarations for port types.
portTypeDefinitions	Generates C constants for port types.
proxyBuilderDeclarations	Generates declarations for functions to create typed proxies.
proxyBuilderImplementations	Generates functions to create typed proxies.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.
relationshipMetadataDeclaration	Generates a forward declaration for the hosting metadata specified in the hostMetadata element.
relationshipMetadataDefinition	Generates a C constant definition for the hosting metadata specified in the hostMetadata element.
structDeclarations	Generates C structure declarations for known types.
structDefinitions	Generates C structure definitions for known types.
stubDeclarations	Generates declarations for stub functions for port type operations.
stubDefinitions	Generates implementations for stub functions for port type operations.
subscriptionFunctionDeclarations	Generates implementation declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionIdlFunctionDeclarations	Generates IDL declarations for subscribe/unsubscribe proxy functions for port type notification operations.

Element	Description
subscriptionProxyFunctionImplementations	Generates implementations for subscribe/unsubscribe proxy functions for port type notification operations.
text	Text and CDATA sections are copied to the file without modification. Source code that is not a function of the contract input data can be added to output files using text and CDATA sections.
thisModelMetadataDeclaration	Generates a forward declaration for the C constant for the manufacturer metadata specified in the thisModelMetadata element.
thisModelMetadataDefinition	Generates a C constant for the manufacturer metadata specified in the thisModelMetadata element.
typeTableDeclarations	Generates C constant declarations for XML schema tables for known types.
typeTableDefinitions	Generates C constants for XML schema tables for known types.

Child element sequence

syntax

```
(
  text,
  CDATA,
  namespaceDeclarations*,
  namespaceDefinitions*,
  structDeclarations*,
  structDefinitions*,
  typeTableDeclarations*,
  typeTableDefinitions*,
  thisModelMetadataDeclaration*,
  thisModelMetadataDefinition*,
  portTypeDeclarations*,
  portTypeDefinitions*,
  messageStructureDefinitions*,
  messageTypeDeclarations*,
```

```

messageTypeDefinitions*,
idlFunctionDeclarations*,
subscriptionIdlFunctionDeclarations*,
functionDeclarations*,
subscriptionFunctionDeclarations*,
proxyFunctionImplementations*,
subscriptionProxyFunctionImplementations*,
stubDeclarations*,
stubDefinitions*,
enumerationValueDeclarations*,
include*,
IUnknownDeclarations*,
IUnknownDefinitions*,
relationshipMetadataDeclaration*,
relationshipMetadataDefinition*,
proxyBuilderDeclarations*,
proxyBuilderImplementations*,
hostBuilderDeclaration*,
hostBuilderImplementation*,
eventSourceBuilderDeclarations*,
eventSourceBuilderImplementations*,
literalInclude*
)

```

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

The name of the file is determined by the value of the name attribute or child element. The content of the file is determined by the other child elements, text and CDATA in the **file** element. Text and CDATA are copied to the file unmodified. Child elements are replaced with generated code. Text, CDATA and child elements may occur in any order and may be repeated indefinitely.

Element information

Label	Value
Minimum supported system	Windows Vista

Label	Value
Can be empty	No

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

functionDeclarations element

Article • 04/26/2021

Generates implementation declarations for proxy functions for port type operations.

Usage

syntax

```
<functionDeclarations>  
  child elements  
</functionDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
async	Specifies whether asynchronous operations are included in the generated proxy functions.
events	Specifies whether related events are included in the generated functions.
faultInfo	Specifies whether parameters used to pass fault information are included in generated functions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(  
    portType?,  
    operation*,  
    events?,  
    async?,  
    faultInfo?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

This element generates declarations of member functions corresponding to operations called out by the contract. These declarations are in a form appropriate for consumption by a C++ compiler and are generally used in .cpp files.

Example:

```
syntax
```

```
<functionDeclarations events = "true"/>
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

host element

Article • 04/26/2021

Defines the [ServiceID](#) and [Types](#) elements for the service host.

Usage

syntax

```
<host>
  child elements
</host>
```

Attributes

There are no attributes.

Child elements

Element	Description
ServiceID	Defines the service identifier for the service host.
Types	Defines a list of XSD qualified names.

Child element sequence

syntax

```
(
  ServiceID,
  Types
)
```

Parent elements

Element	Description

Element	Description
hostMetadata	The hosting metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

hostBuilderDeclaration element

Article • 04/26/2021

Generates a declaration for a function that creates a typed host.

Usage

syntax

```
<hostBuilderDeclaration>  
  child elements  
</hostBuilderDeclaration>
```

Attributes

There are no attributes.

Child elements

Element	Description
interface	The name of service interface to be included for host. The value of this element must match the value of the interface child element of hostedService .

Child element sequence

syntax

```
interface+
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

hostBuilderImplementation element

Article • 04/26/2021

Generates a function that creates a typed host.

Usage

syntax

```
<hostBuilderImplementation>  
  child elements  
</hostBuilderImplementation>
```

Attributes

There are no attributes.

Child elements

Element	Description
hostedService	The service to be included for the host.

Child element sequence

syntax

```
hostedService+
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

hosted element

Article • 04/26/2021

Defines the [ServiceID](#), [Types](#), [PnP-XHardwareId](#), and [PnP-XCompatibleId](#) elements for the services defined by the service host.

Usage

syntax

```
<hosted>
  child elements
</hosted>
```

Attributes

There are no attributes.

Child elements

Element	Description
PnP-XCompatibleId	Specifies the PnP-X Compatible Identifier of the service.
PnP-XHardwareId	Specifies the PnP-X Hardware Identifier of the service.
ServiceID	Defines a service identifier for the service host.
Types	Defines a list of XSD qualified names.

Child element sequence

syntax

```
(
  ServiceID,
  Types,
  PnP-XHardwareId?,
```

```
PnPCompatibleId?  
)
```

Parent elements

Element	Description
hostMetadata	The hosting metadata for the device to be implemented.

Remarks

Each service provided by a service host should have its own **hosted** element information to ensure that the service is published properly in responses to metadata requests.

The [PnPHardwareId](#) and [PnPCompatibleId](#) elements are optional, but when used, they must be used together. If one is present, the other must be present as well.

If a [PnPDeviceCategory](#) element is present, then at least one **hosted** element must contain both [PnPHardwareId](#) and [PnPCompatibleId](#) elements. Similarly, if [PnPHardwareId](#) and [PnPCompatibleId](#) elements are present in a **hosted** element, at least one [PnPDeviceCategory](#) element must be present inside the [thisModelMetadata](#) element.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

hostedService element

Article • 04/26/2021

Defines a service to be hosted in a host builder function.

Usage

syntax

```
<hostedService>  
  child elements  
</hostedService>
```

Attributes

There are no attributes.

Child elements

Element	Description
codeName	Specifies the name to be used for the port type in the generated code. By default, the code name is generated from the port type's qualified name.
portType	Port type for which code is to be generated.
proxyClass	Class name to generate from builder function.
ServiceID	A URI that represents the service identifier.

Child element sequence

syntax

```
codeName(  
  ServiceID,  
  proxyClass,
```

```
portType+  
)
```

Parent elements

Element	Description
file	The output file specification for the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

hostMetadata element

Article • 04/26/2021

Defines the hosting metadata for the device to be implemented. This element is used for device implementations (hosts) only.

Usage

syntax

```
<hostMetadata>
  child elements
</hostMetadata>
```

Attributes

There are no attributes.

Child elements

Element	Description
host	Defines the ServiceID and Types elements for the service host. If not provided explicitly, WSDAPI will not supply any default data in response to metadata requests.
hosted	Defines the ServiceID and Types elements for the services provided by this service host. Each service provided by this service host should have its own hosted element information to ensure that the service is published properly in responses to metadata requests.

Child element sequence

syntax

```
(
  host?,
  hosted+
)
```

Parent elements

Element	Description
relationshipMetadata	Describes the host and hosted metadata for the device.

Remarks

The hosting metadata appears in this element in a form similar to the one specified in the device profile. **hostMetadata** is slightly different from the format described in the device profile in that the only reference property it supports is the service ID.

The [relationshipMetadataDefinition](#) element is used subsequently to generate a C constant containing this information.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

idlFunctionDeclarations element

Article • 04/26/2021

Generates IDL declarations for proxy functions for port type operations.

Usage

syntax

```
<idlFunctionDeclarations>  
  child elements  
</idlFunctionDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
async	Specifies whether asynchronous operations are included in the generated proxy functions.
eventArg	Specifies whether related event arguments are included in the generated functions.
events	Specifies whether related events are included in the generated functions.
faultInfo	Specifies whether parameters used to pass fault information are included in generated functions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(  
  portType?,  
  operation*,  
  eventArg?,  
  events?,  
  async?,  
  faultInfo?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

This element generates declarations of member functions corresponding to operations called out by the contract. These declarations are in a form appropriate for consumption by the MIDL compiler and are generally used in .idl files.

Example:

syntax

```
<idlFunctionDeclarations events = "true"/>
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

importHint element

Article • 04/26/2021

Specifies the download location for a `<wsdl:import>` directive that does not explicitly specify a location.

Usage

syntax

```
<importHint>
  child elements
</importHint>
```

Attributes

There are no attributes.

Child elements

Element	Description
location	The location of the file to import. The location may be a relative path, an absolute path, or an HTTP URL.
namespace	The namespace to import. This should match the namespace specified in the <code><wsdl:import></code> element.

Child element sequence

syntax

```
(
  namespace,
  location
)
```

Parent elements

Element	Description
wsdl	Specifies a WSDL file to process for contract information.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

include element

Article • 04/26/2021

Includes the contents of a macro or file in the generated output.

Usage

syntax

```
<include  
  macro = "character_string"  
  file = "character_string"/>
```

Attributes

Attribute	Type	Required	Description
file	character_string	No	The path to the file to include.
macro	character_string	No	The name of the macro to include.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Directs the code generator to generate a file and specifies the output file name.

Remarks

Either the **macro** attribute or the **file** attribute must be specified. Do not specify both attributes.

WsdCodeGen defines a macro named **DoNotModify**. When this macro is included, the generated code contains a high-visibility comment block that instructs developers not to modify the generated code.

Examples

The following XML shows how to include the **DoNotModify** macro. This XML can be added to an XML configuration file for WsdCodeGen.

syntax

```
<include macro="DoNotModify">
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

interface element

Article • 04/26/2021

The name of the interface to be returned by QueryInterface.

Usage

syntax

```
<interface/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
IUnknownDefinitions	Generates implementations for the QueryInterface, AddRef and Release functions.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

IUnknownDeclarations element

Article • 04/26/2021

Generates declarations for the QueryInterface, AddRef and Release functions.

Usage

syntax

```
<IUnknownDeclarations/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

IUnknownDefinitions element

Article • 04/26/2021

Generates implementations for the QueryInterface, AddRef and Release functions.

Usage

syntax

```
<IUnknownDefinitions
  proxyClass = "character_string"
  refCountVar = "character_string">
  child elements
</IUnknownDefinitions>
```

Attributes

Attribute	Type	Required	Description
proxyClass	character_string	Yes	The name of the implementing class.
refCountVar	character_string	Yes	The reference count variable name.

Child elements

Element	Description
interface	The name of the interface to be returned by QueryInterface.

Child element sequence

syntax

interface

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

layerNumber element

Article • 04/26/2021

Defines the number of the code layer to be generated.

Usage

syntax

```
<layerNumber/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

Layer numbers are used in runtime tables to distinguish one layer of code for another. WSDAPI itself uses generated code that has a layer number of 0.

Typically, this value will be 1, indicating that the generated code is layered on top of WSDAPI and no other layer. In some cases, higher numbers may be used. For example, if one company produces code for a device class (numbered layer 1), a particular hardware vendor may want to add an additional layer numbered layer 2.

Legal values, except in the case of WSDAPI itself are greater than 0 and less than 16.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

layerPrefix element

Article • 04/26/2021

Defines the prefix to use in the generated code to assure uniqueness of generated symbols.

Usage

syntax

```
<layerPrefix/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

Each code generator script should define a unique prefix string for the modules created. For example, the Picture Frame scripts use a prefix of "PFDEMO".

WSDAPI uses the prefix "WSD".

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

literalInclude element

Article • 08/20/2021

Places a C or IDL include statement in the generated code.

Usage

syntax

```
<literalInclude  
  Language = "language string"  
  Local = "Boolean"/>
```

Attributes

Attribute	Type	Required	Description
Language	language string	No	The type of header file to be included. C Include a C header file. IDL Include an IDL file.
Local	Boolean	No	This attribute is only used when Language is set to "C". True Searches the current directory for the named header before searching the system directories. False Only search system directories for the named header.

Child elements

There are no child elements.

Parent elements

Element	Description

Element	Description
file	Outputs a file from the code generator.

Remarks

The following examples show the code generated from different **literalInclude** elements.

Example 1 - Generating a C include statement

Input XML:

```
syntax
```

```
<literalInclude Language="C" Local="False">wsdapi.h</literalInclude>
```

Output code:

```
syntax
```

```
#include <wsdapi.h>
```

Example 2 - Generating a C include statement

Input XML:

```
syntax
```

```
<literalInclude Language="C" Local="True">wsdapi.h</literalInclude>
```

Output code:

```
syntax
```

```
#include "wsdapi.h"
```

Example 3 - Generating an IDL import statement

Input XML:

syntax

```
<literalInclude Language="IDL">wsdclient.idl</literalInclude>
```

Output code:

syntax

```
import wsdclient.idl;
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

location element

Article • 04/26/2021

Specifies the URL provided as an import hint for a WSDL or XSD referenced inside another WSDL.

Usage

syntax

```
<location/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
importHint	Specifies the download location for a <wsdl:import> directive that does not explicitly specify a location.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

macro element

Article • 04/26/2021

Defines text or CDATA to be reused by the [include](#) element.

Usage

syntax

```
<macro  
  name = "character_string"/>
```

Attributes

Attribute	Type	Required	Description
name	character_string	Yes	The name of the macro.

Child elements

There are no child elements.

Parent elements

Element	Description
WsdCodeGen	The root element of a WSDAPI code generator XML script file.

Remarks

WsdCodeGen defines a macro named **DoNotModify**. When this macro is included, the generated code contains a high-visibility comment block that instructs developers not to modify the generated code.

The following XML shows how to include the **DoNotModify** macro. This XML can be added to an XML configuration file for WsdCodeGen.

syntax

```
<include macro="DoNotModify">
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

macroPrefix element

Article • 04/26/2021

Defines the prefix to be used in the generated code for names of macros in the namespace.

Usage

syntax

```
<macroPrefix/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
nameSpace	A namespace to be used for code generation.

Remarks

This element overrides the default URI prefix used for generated macros. For example, if the macro prefix is "AV_" and the name is "Tuner", the macro generated for the qualified name will be "AV_Tuner".

By default, the code generated creates a preferred macro prefix from the URI.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

manufacturer element

Article • 04/26/2021

Specifies the name of the manufacturer.

Usage

syntax

```
<manufacturer/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

manufacturerLS element

Article • 04/26/2021

Specifies a localized version of the manufacturer name.

Usage

syntax

```
<manufacturerLS  
  Language = "language identifier string"  
  Data = "localized manufacturer name string"/>
```

Attributes

Attribute	Type	Required	Description
Data	localized manufacturer name string	Yes	The localized manufacturer name.
Language	language identifier string	Yes	The language of the localized manufacturer name.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

manufacturerURL element

Article • 04/26/2021

Specifies the URL address for the manufacturer.

Usage

syntax

```
<manufacturerURL/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

messageStructureDefinitions element

Article • 04/26/2021

Generates C structure definitions for message types.

Usage

syntax

```
<messageStructureDefinitions>
  child elements
</messageStructureDefinitions>
```

Attributes

There are no attributes.

Child elements

Element	Description
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(
  portType?,
  operation*
)
```

Parent elements

Element	Description

Element	Description
file	Outputs a file from the code generator.

Remarks

Message structures are referenced by generated proxy and stub code. This element is used to generate include files.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

messageTypeDeclarations element

Article • 04/26/2021

Generates C constant declarations for XML schema tables for message types.

Usage

syntax

```
<messageTypeDeclarations>
  child elements
</messageTypeDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(
  portType?,
  operation*
)
```

Parent elements

Element	Description

Element	Description
file	Outputs a file from the code generator.

Remarks

Schema tables are referenced by port type definitions. This element is therefore generally used just prior to [portTypeDefinitions](#) elements.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

messageTypeDefinitions element

Article • 04/26/2021

Generates C constants for XML schema tables for message types.

Usage

syntax

```
<messageTypeDefinitions>  
  child elements  
</messageTypeDefinitions>
```

Attributes

There are no attributes.

Child elements

Element	Description
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(  
  portType?,  
  operation*  
)
```

Parent elements

Element	Description

Element	Description
file	Outputs a file from the code generator.

Remarks

This element is generally used in C source files to provide the schema tables declared by [messageTypeDeclarations](#).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

modelName element

Article • 04/26/2021

Specifies the name of the device.

Usage

syntax

```
<modelName/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

modelNameLS element

Article • 04/26/2021

Specifies a localized version of the device name.

Usage

syntax

```
<modelNameLS  
  Language = "language identifier string"  
  Data = "localized model name string"/>
```

Attributes

Attribute	Type	Required	Description
Data	localized model name string	Yes	The localized model name.
Language	language identifier string	Yes	The language of the localized model name.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

modelName element

Article • 04/26/2021

Specifies the model number of the device.

Usage

syntax

```
<modelName/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

modelURL element

Article • 04/26/2021

Specifies the URL address for the device model.

Usage

syntax

```
<modelURL/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

name element

Article • 04/26/2021

Specifies a qualified name in the namespace.

Usage

syntax

```
<name/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
nameSpace	A namespace to be used for code generation.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 yes

 no

[Get help at Microsoft Q&A](#)

nameSpace element

Article • 04/26/2021

Describes a namespace. This namespace may be used for code generation or for handling `<wsdl:import>` directives.

Usage

syntax

```
<nameSpace
  uri = "character_string">
  child elements
</nameSpace>
```

Attributes

Attribute	Type	Required	Description
uri	character_string	Yes	The unique URI of the namespace.

Child elements

Element	Description
codeName	The name to be used to identify the namespace in generated code.
macroPrefix	The prefix to be used in the generated code for names of macros in the namespace.
name	A qualified name in the namespace.
preferredPrefix	The prefix to which the namespace should be mapped to make the XML more readable.

Child element sequence

syntax

```
(
  preferredPrefix?,
  macroPrefix?,
  codeName?,
  name*
)
```

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

This element may be used to provide the code generator with additional information about namespaces that it encounters in WSDL and XSD files or to introduce new namespaces.

Namespaces implied by type reflection or specified in WSDL and XSD files are parsed automatically by the code generator and do not have to be explicitly defined in the script. In some cases, this element can be used to add additional properties or names to a namespace that is referenced by type reflection or WSDL/XSD. The code generator does not regard this as a conflict.

The following XML shows a nameSpace element (with children omitted for brevity).

syntax

```
<nameSpace uri="https://www.example.com/namespace">
  ...
</nameSpace>
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

namespaceDeclarations element

Article • 04/26/2021

Generates C declarations for namespace tables.

Usage

syntax

```
<namespaceDeclarations/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

Namespace tables are referenced by type tables and other generated code, so this element is used to generate include files.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

namespaceDefinitions element

Article • 04/26/2021

Generates C definitions for namespace tables.

Usage

syntax

```
<namespaceDefinitions/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

This element is generally used in C source files to provide the namespace tables which were declared by [namespaceDeclarations](#).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

notificationInterface element

Article • 04/26/2021

Specifies the name of the notification interface used with event subscriptions.

Usage

syntax

```
<notificationInterface/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
subscriptionFunctionDeclarations	Generates implementation declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionIdlFunctionDeclarations	Generates IDL declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionProxyFunctionImplementations	Generates implementations for subscribe/unsubscribe proxy functions for port type notification operations.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

operation element

Article • 04/26/2021

Specifies an operation for which code is to be generated.

Usage

syntax

```
<operation/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
functionDeclarations	Generates implementation declarations for proxy functions for port type operations.
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
messageStructureDefinitions	Generates C structure definitions for message types.
messageTypeDeclarations	Generates C constant declarations for XML schema tables for message types.
messageTypeDefinitions	Generates C constants for XML schema tables for message types.

Element	Description
portTypeDeclarations	Generates C constant declarations for port types.
portTypeDefinitions	Generates C constants for port types.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.
stubDeclarations	Generates declarations for stub functions for port type operations.
stubDefinitions	Generates implementations for stub functions for port type operations.
subscriptionFunctionDeclarations	Generates implementation declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionIdlFunctionDeclarations	Generates IDL declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionProxyFunctionImplementations	Generates implementations for subscribe/unsubscribe proxy functions for port type notification operations.

Remarks

Any number of operations may be specified. If no operations are specified, code is generated for all operations in all the relevant port types. Using the **operation** element will limit the methods generated to those contained in the operation.

For example, a printer supports these operations among others:

- **PrintJobByPost**
- **PrintJobByReference**
- **CancelJob**
- **GetJobElements**
- **GetActiveJobs**

- **GetJobHistory**
- **SubscribeToPrinterConfigChange**
- **UnsubscribeToPrinterConfigChange**

However, to include only the methods related to the **PrintJobByPost** and **GetJobElements** operations, the code generation script would use the **idlFunctionDeclarations** elements as follows:

syntax

```
<idlFunctionDeclarations>
  <operation>PrintJobByPost</operation>
  <operation>GetJobElements<</operation>
</idlFunctionDeclarations>
```

This generates idl function declarations for all methods associated with the two operations (for example, **BeginPrintJobByPost**, **EndPrintJobByPost**, **BeginGetJobElements** and **EndGetJobElements**).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

operationDeallocator element

Article • 04/26/2021

Specifies the type deallocator to be used by an operation's stub function.

Usage

syntax

```
<operationDeallocator  
  operation = "character_string"  
  deallocator = "character_string"/>
```

Attributes

Attribute	Type	Required	Description
deallocator	character_string	Yes	The deallocator to use for the operation. The following strings are valid values. none WSDFreeLinkedMemory CoTaskMemFree free delete deleteArray Release
operation	character_string	Yes	The name of the operation.

Child elements

There are no child elements.

Parent elements

Element	Description

Element	Description
stubDefinitions	Generates implementations for stub functions for port type operations.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

path element

Article • 04/26/2021

Specifies the location and filename of a WSDL file. The path may be an absolute or relative path to the file, but must not be an URL.

Usage

syntax

```
<path/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
wsdl	A WSDL file to process for contract information.

Examples

The following XML shows a valid **path** element.

syntax

```
<path>"..\wsdl\example.wsdl"</path>
```

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

PnPXCompatibleId element

Article • 04/26/2021

Specifies the PnP-X Compatible Identifier of the service. Devices may have more than one compatible ID.

Usage

syntax

```
<PnPXCompatibleId/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
hosted	Defines elements for the services defined by the service host.

Remarks

To specify more than one CompatibleID, separate the identifiers with a space character, for example, "PnPX_SampleService_HWID_1 PnPX_SampleService_HWID_2 PnPX_SampleService1_HWID_3".

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

PnPDeviceCategory element

Article • 04/26/2021

Specifies the PnP-X category to which the device belongs. More than one category can be specified.

Usage

syntax

```
<PnPDeviceCategory/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Remarks

Devices can also specify a device subcategory for a more descriptive device category. For example, "Phones.WindowsMobile Cameras.DigitalStillCamera MediaDevices.MusicPlayer" identifies a device that is a Microsoft Windows Mobile device with a camera and music player. The primary device category for this device would be "Phones".

To specify more than one device category, separate the categories with a space. For example, "Printers Storage" identifies a device with a primary category of "Printers" and

a secondary category of "Storage".

If a **PnPDeviceCategory** element is present, then at least one **hosted** element should contain both **PnPHardwareId** and **PnPCompatibleId** elements. Similarly, if **PnPHardwareId** and **PnPCompatibleId** elements are present in a **hosted** element, at least one **PnPDeviceCategory** element should be present inside the **thisModelMetadata** element.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

PnPXHardwareId element

Article • 04/26/2021

Specifies the PnP-X Hardware Identifier of the service. PnP matches this element with the hardware description(s) provided in the [PnpDevice] section of the device's INF file. Based on this information, the PnP service selects and loads the most appropriate device driver.

Usage

syntax

```
<PnPXHardwareId/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
hosted	Defines elements for the services defined by the service host.

Remarks

To specify more than one hardware ID, separate the identifiers with a space character, for example, "PnPX_SampleService_HWID_1 PnPX_SampleService_HWID_2 PnPX_SampleService1_HWID_3".

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

portType element

Article • 04/26/2021

Specifies the port type for which code is to be generated.

Usage

syntax

```
<portType/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
functionDeclarations	Generates implementation declarations for proxy functions for port type operations.
idlFunctionDeclarations	Generates IDL declarations for proxy functions for port type operations.
messageStructureDefinitions	Generates C structure definitions for message types.
messageTypeDeclarations	Generates C constant declarations for XML schema tables for message types.
messageTypeDefinitions	Generates C constants for XML schema tables for message types.

Element	Description
portTypeDeclarations	Generates C constant declarations for port types.
portTypeDefinitions	Generates C constants for port types.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.
stubDeclarations	Generates declarations for stub functions for port type operations.
stubDefinitions	Generates implementations for stub functions for port type operations.
subscriptionFunctionDeclarations	Generates implementation declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionIdlFunctionDeclarations	Generates IDL declarations for subscribe/unsubscribe proxy functions for port type notification operations.
subscriptionProxyFunctionImplementations	Generates implementations for subscribe/unsubscribe proxy functions for port type notification operations.

Remarks

By default, code is generated for all port types in WSDL input files.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

portTypeDeclarations element

Article • 04/26/2021

Generates C constant declarations for port types.

Usage

syntax

```
<portTypeDeclarations>
  child elements
</portTypeDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
codeName	Specifies the name to be used for the port type in the generated code. By default, the code name is generated from the port type's qualified name.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(
  portType?,
  operation*,
  codeName?
)
```


Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

Port type declarations are referenced by the application and much of the generated code. This element is used to generate include files.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

portTypeDefinitions element

Article • 04/26/2021

Generates C constants for port types.

Usage

syntax

```
<portTypeDefinitions>  
  child elements  
</portTypeDefinitions>
```

Attributes

There are no attributes.

Child elements

Element	Description
codeName	Specifies the name to be used for the port type in the generated code. By default, the code name is generated from the port type's qualified name.
eventStubFunction	Specifies whether stub function references should be included in the operation structures in the port type definitions for notification operations.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.
stubFunction	Specifies whether stub function references should be included in the operation structures in the port type definitions for one-way and two-way operations.

Child element sequence

syntax

```
(  
    portType?,  
    operation*,  
    stubFunction?,  
    eventStubFunction?,  
    codeName?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

This element is generally used in C source files to provide the port type constants declared by [portTypeDeclarations](#).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

preferredPrefix element

Article • 04/26/2021

Defines the prefix to which the namespace should be mapped to make the XML more readable.

Usage

syntax

```
<preferredPrefix/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
nameSpace	A namespace to be used for code generation.

Remarks

This element overrides the default URI prefix used for generated code. For example, a media-related namespace might have the preferred prefix "av" (for audio/visual).

By default, the code generated creates a preferred prefix from the URI.

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

presentationURL element

Article • 04/26/2021

Specifies the URL address of the device model's presentation data.

Usage

syntax

```
<presentationURL/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
thisModelMetadata	Defines the manufacturer and model metadata for the device to be implemented.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

proxyBuilderDeclarations element

Article • 04/26/2021

Generates declarations for functions that create typed proxies.

Usage

syntax

```
<proxyBuilderDeclarations>  
  child elements  
</proxyBuilderDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
proxyClass	The class name to generate from the proxy builder function.

Child element sequence

syntax

```
proxyClass
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

proxyBuilderImplementations element

Article • 04/26/2021

Generates functions to create typed proxies.

Usage

syntax

```
<proxyBuilderImplementations>  
  child elements  
</proxyBuilderImplementations>
```

Attributes

There are no attributes.

Child elements

Element	Description
codeName	The name to be used for the port type in the generated code. By default, the code name is generated from the port type's qualified name.
portType	Port type for which code is to be generated.
proxyClass	Class name to generate from the proxy builder function.

Child element sequence

syntax

```
(  
  proxyClass,  
  portType+,  
  codeName  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

proxyClass element

Article • 04/26/2021

Specifies the name of the client-side proxy class.

Usage

syntax

```
<proxyClass/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
eventSourceBuilderDeclarations	Generates declarations for functions that create event source classes.
eventSourceBuilderImplementations	Generates functions that create event source classes.
proxyFunctionImplementations	Generates implementations for proxy functions for port type operations.
subscriptionProxyFunctionImplementations	Generates implementations for subscribe/unsubscribe proxy functions for port type notification operations.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

proxyFunctionImplementations element

Article • 04/26/2021

Generates implementations for proxy functions for port type operations.

Usage

syntax

```
<proxyFunctionImplementations>  
  child elements  
</proxyFunctionImplementations>
```

Attributes

There are no attributes.

Child elements

Element	Description
async	Specifies whether asynchronous operations are included in the generated proxy functions.
events	Specifies whether related events are included in the generated functions.
faultInfo	Specifies whether parameters used to pass fault information are included in generated functions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.
proxyClass	Specifies the name of the client-side proxy class.

Child element sequence

syntax

```
(  
  portType?,  
  operation*,  
  proxyClass?,  
  events?,  
  async?,  
  faultInfo?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

relationshipMetadata element

Article • 04/26/2021

Describes the host and hosted metadata for the device.

Usage

syntax

```
<relationshipMetadata/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

relationshipMetadataDeclaration element

Article • 04/26/2021

Generates a forward declaration for the hosting metadata specified in the [hostMetadata](#) element. This declaration is used in a header file.

Usage

syntax

```
<relationshipMetadataDeclaration/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

Hosting metadata is referenced by the application when it initializes the host. This element is used to generate include files that are included by the application source code.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

relationshipMetadataDefinition element

Article • 04/26/2021

Generates a C constant definition for the hosting metadata specified in the [hostMetadata](#) element. This definition is used in a source file.

Usage

syntax

```
<relationshipMetadataDefinition/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

This element is generally used in C source files to provide the service host metadata declared by [relationshipMetadataDeclaration](#).

Element information

Label	Value
Minimum supported system	Windows Vista

Label	Value
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

serverClass element

Article • 04/26/2021

Specifies the name of the host-side server class.

Usage

syntax

```
<serverClass/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
stubDefinitions	Generates implementations for stub functions for port type operations.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 yes

 no

[Get help at Microsoft Q&A](#)

ServiceID element

Article • 04/26/2021

A URI that represents the service identifier.

Usage

syntax

```
<ServiceID/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
host	Defines the ServiceID and Types elements for the service host. If not provided explicitly, WSDAPI will not supply any default data in response to metadata requests.
hosted	Defines the ServiceID and Types elements for the services provided by this service host. Each service provided by this service host should have its own hosted element information to ensure that the service is published properly in responses to metadata requests.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

structDeclarations element

Article • 04/26/2021

Generates C structure declarations for known types.

Usage

syntax

```
<structDeclarations/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

Structures for known types are referenced by much of the generated code and by application code. This element is used to generate include files.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

structDefinitions element

Article • 04/26/2021

Generates C structure definitions for known types.

Usage

syntax

```
<structDefinitions/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

Structures for known types are referenced by much of the generated code and by application code. This element is used to generate include files. This element should occur after a [structDeclarations](#) element so that references between structures are handled properly.

Element information

Label	Value

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

stubDeclarations element

Article • 04/26/2021

Generates declarations for stub functions for port type operations.

Usage

syntax

```
<stubDeclarations>  
  child elements  
</stubDeclarations>
```

Attributes

There are no attributes.

Child elements

Element	Description
events	Specifies whether related events are included in the generated functions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(  
  portType?,  
  operation*,  
  events?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

stubDefinitions element

Article • 04/26/2021

Generates implementations for stub functions for port-type operations.

Usage

syntax

```
<stubDefinitions>  
  child elements  
</stubDefinitions>
```

Attributes

There are no attributes.

Child elements

Element	Description
deallocator	Specifies the type of deallocator to be used by a stub function.
eventArg	Specifies whether related event arguments are included in the generated functions.
events	Specifies whether related events are included in the generated functions.
faultInfo	Specifies whether parameters used to pass fault information are included in generated functions.
operation	Specifies an operation for which code is to be generated.
operationDeallocator	Specifies the type of deallocator to be used by an operation's stub function.
portType	Specifies the port type for which code is to be generated.

Element	Description
serverClass	Specifies the name of the host-side server class.

Child element sequence

syntax
<pre>(portType?, operation*, serverClass, eventArg?, events?, operationDeallocator*, deallocator?, faultInfo?)</pre>

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

stubFunction element

Article • 04/26/2021

Specifies whether stub function references should be included in the operation structures in the port type definitions for one-way and two-way operations.

Usage

syntax

```
<stubFunction/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
portTypeDefinitions	Generates C constants for port types.

Remarks

Stub function references are used in hosting scenarios for one-way and two-way operations.

Valid values for this element are 1 (TRUE/stub function references included) and 0 (FALSE/no stub function references included).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

subscriptionFunctionDeclarations element

Article • 04/26/2021

Generates implementation declarations for subscribe/unsubscribe proxy functions for port type notification operations.

Usage

syntax

```
<subscriptionFunctionDeclarations
  extensible = "boolean">
  child elements
</subscriptionFunctionDeclarations>
```

Attributes

Attribute	Type	Required	Description
extensible	boolean	No	The ability to add extensibility points to functions and interfaces. This value is always set to true.

Child elements

Element	Description
notificationInterface	Specifies the name of the notification interface used with event subscriptions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(  
  portType?,  
  operation*,  
  notificationInterface?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

subscriptionIdlFunctionDeclarations element

Article • 04/26/2021

Generates IDL declarations for subscribe/unsubscribe proxy functions for port type notification operations.

Usage

syntax

```
<subscriptionIdlFunctionDeclarations  
  extensible = "boolean">  
  child elements  
</subscriptionIdlFunctionDeclarations>
```

Attributes

Attribute	Type	Required	Description
extensible	boolean	No	The ability to add extensibility points to functions and interfaces. This value is always set to true.

Child elements

Element	Description
notificationInterface	Specifies the name of the notification interface used with event subscriptions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.

Child element sequence

syntax

```
(  
  portType?,  
  operation*,  
  notificationInterface?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

subscriptionProxyFunctionImplementations element

Article • 04/26/2021

Generates implementations for subscribe/unsubscribe proxy functions for port type notification operations.

Usage

syntax

```
<subscriptionProxyFunctionImplementations
  extensible = "boolean">
  child elements
</subscriptionProxyFunctionImplementations>
```

Attributes

Attribute	Type	Required	Description
extensible	boolean	No	The ability to add extensibility points to functions and interfaces. This value is always set to true.

Child elements

Element	Description
notificationInterface	Specifies the name of the notification interface used with event subscriptions.
operation	Specifies an operation for which code is to be generated.
portType	Specifies the port type for which code is to be generated.
proxyClass	Specifies the name of the client-side proxy class.

Child element sequence

syntax

```
(  
  portType?,  
  operation*,  
  proxyClass?,  
  notificationInterface?  
)
```

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

thisModelMetadata element

Article • 04/26/2021

Defines the manufacturer and model metadata for the device to be implemented. This element is used for device implementations (hosts) only.

Usage

syntax

```
<thisModelMetadata>
  child elements
</thisModelMetadata>
```

Attributes

There are no attributes.

Child elements

Element	Description
manufacturer	Name of the manufacturer. At least one of manufacturer or manufacturerLS must be present in the metadata.
manufacturerLS	Localized version of the manufacturer name.
manufacturerURL	Manufacturer URL address.
modelName	Name of the device. At least one of modelName or modelNameLS must be present in the metadata.
modelNameLS	Localized version of the device name.
modelNumber	Model number of the device.

Element	Description
modelURL	URL address for the device model.
PnPXDeviceCategory	PnP-X category to which the device belongs.
presentationURL	URL address of the device model's presentation data.

Child element sequence

syntax

```
(
  manufacturer?,
  manufacturerLS*,
  manufacturerURL?,
  modelName?,
  modelNameLS*,
  modelNumber,
  modelURL?,
  PnPXDeviceCategory?,
  presentationURL?
)
```

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

The manufacturer metadata matches the manufacturer metadata section described in the device profile (consult the device profile for details). The manufacturer name or at least one localized version of the manufacturer name must be provided. The model name or at least one localized version of the model name must be provided.

The [thisModelMetadataDefinition](#) element is subsequently used to generate a C constant containing this information.

If a **PnPXDeviceCategory** element is present, then at least one **hosted** element must contain both **PnPXHardwareId** and **PnPXCompatibleId** elements. Similarly, if **PnPXHardwareId** and **PnPXCompatibleId** elements are present in a **hosted** element, a **PnPXDeviceCategory** element must be present inside the **thisModelMetadata** element.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

thisModelMetadataDeclaration element

Article • 04/26/2021

Generates a forward declaration for the C constant for the manufacturer metadata specified in the [thisModelMetadata](#) element. This declaration is used in a header file.

Usage

syntax

```
<thisModelMetadataDefinition/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

Get help at [Microsoft Q&A](#)

thisModelMetadataDefinition element

Article • 04/26/2021

Generates a C constant for the manufacturer metadata specified in the [thisModelMetadata](#) element. This definition is used in a source file.

Usage

syntax

```
<thisModelMetadataDefinition/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

Get help at [Microsoft Q&A](#)

Types element

Article • 04/26/2021

A list of XSD qualified names.

Usage

syntax

<Types/>

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
host	Identifies the service host.
hosted	Identifies a service defined by a service host.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

typeTableDeclarations element

Article • 04/26/2021

Generates C constant declarations for XML schema tables for known types.

Usage

syntax

```
<typeTableDeclarations/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

XML schema tables are referenced by much of the generated code. This element is used to generate include files.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

typeTableDefinitions element

Article • 04/26/2021

Generates C constants for XML schema tables for known types.

Usage

syntax

```
<typeTableDefinitions/>
```

Attributes

There are no attributes.

Child elements

There are no child elements.

Parent elements

Element	Description
file	Outputs a file from the code generator.

Remarks

This element is generally used in C source files to provide the XML schema tables declared by [typeTableDeclarations](#).

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

typeUri element

Article • 04/26/2021

Specifies a type to include from an XSD file.

Usage

syntax

```
<typeUri
  type = "character_string"
  uri = "character_string"/>
```

Attributes

Attribute	Type	Required	Description
type	character_string	Yes	The name of the type.
uri	character_string	Yes	The namespace of the type. Must be a valid URI.

Child elements

There are no child elements.

Parent elements

Element	Description
xsd	Specifies an XSD file to process for contract information.

Element information

Label	Value
Minimum supported system	Windows Vista

Label	Value
Can be empty	Yes

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

wsdCodeGen element

Article • 04/26/2021

Is the root element of an WSDAPI code generator XML script file.

Usage

syntax

```
<wsdCodeGen
  ConfigFileVersion = "Any character string.">
  child elements
</wsdCodeGen>
```

Attributes

Attribute	Type	Required	Description
ConfigFileVersion	Any character string.	Yes	The version of the configuration file. The only valid value is "1.0".

Child elements

Element	Description
autoStatic	Indicates whether or not WsdCodeGen should try to automatically flag certain generated fields as static. This is enabled by default.
file	Directs the code generator to generate a file.
hostMetadata	The hosting metadata for the device to be implemented. This element is used for device implementations (hosts) only.
layerNumber	The number of the code layer to be generated. Layer numbers are used in runtime tables to distinguish one layer of code for another. WSDAPI itself uses generated code that has a layer number of 0.

Element	Description
layerPrefix	The prefix to use in the generated code to assure uniqueness of generated symbols. WSDAPI uses the prefix "WSD".
macro	Defines text or CDATA to be reused by the include element.
nameSpace	Describes a namespace to be used for code generation.
relationshipMetadata	Describes the host and hosted metadata for the device.
thisModelMetadata	The manufacturer and model metadata for the device to be implemented. This element is used for device implementations (hosts) only.
wsdl	Specifies a WSDL file to process for contract information.
xsd	Specifies an XSD file to process for contract information.

Child element sequence

syntax

```
(
  layerNumber?,
  layerPrefix?,
  autoStatic?,
  hostMetadata?,
  thisModelMetadata?,
  nameSpace*,
  wsdl*,
  xsd*,
  file*,
  macro*,
  relationshipMetadata*
)
```

Parent elements

There are no parent elements.

Remarks

In general, [file](#) elements should occur last because they generate code which uses data specified by the other elements.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

wSDL element

Article • 04/26/2021

Specifies a WSDL file to process for contract information.

Usage

syntax

```
<wSDL>
  child elements
</wSDL>
```

Attributes

There are no attributes.

Child elements

Element	Description
excludeImport	Prevents the generation of import statements for specified targets named in a <wSDL:import> element in a WSDL file.
importHint	Specifies the download location for a <wSDL:import> directive that does not explicitly specify a location.
path	File and path of the WSDL input file.

Child element sequence

syntax

```
(
  importHint*,
  excludeImport*,
  path
)
```

Parent elements

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

Any [importHint](#) or [excludeImport](#) elements appearing after the [path](#) element in an XML configuration file will be ignored.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	No

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

xsd element

Article • 04/26/2021

Specifies an XSD file to process for contract information.

Usage

syntax

```
<xsd
  path = "pathname string">
  child elements
</xsd>
```

Attributes

Attribute	Type	Required	Description
path	pathname string	Yes	File and path of the XSD input file.

Child elements

Element	Description
typeUri	Specifies a type to include from an XSD file.

Child element sequence

syntax

```
typeUri*
```

Parent elements

Element	Description

Element	Description
wsdCodeGen	The root element of an WSDAPI code generator XML script file.

Remarks

The filename string should include complete path information as well.

Element information

Label	Value
Minimum supported system	Windows Vista
Can be empty	Yes

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

WSDAPI XML Type Bytecodes

Article • 01/07/2021

These bytecodes are used by WSDAPI to parse and generate XML using bytecode tables which describe complex types.

The bytecode table for a type expresses both the binary (in-memory) and XML (on-the-wire) representations of the type. A bytecode table consists of a list of operations terminated by a special operation, namely `OpEndOfTable`. An operation consists of a one-byte operation code followed by zero or more bytes of argument data.

Bytecode	Description
OpNone	Matches nothing.
OpEndOfTable	Indicates the end of a bytecode table.
OpBeginElement (name)	Matches an element start token with the indicated name. The name appears in the table in 4-byte encoded form. OnBeginElement starts a clause that ends with a matching OpEndElement. Zero or more complete clauses must appear between OpBeginElement and its matching OpEndElement.
OpBeginAnyElement	Matches an element start token with any name. OnBeginAnyElement starts a clause that ends with a matching OpEndElement. Zero or more complete clauses must appear between OpBeginAnyElement and its matching OpEndElement.
OpEndElement	Ends a clause started with OpBeginElement or OpBeginAnyElement.
OpElement (name)	Matches an entire element with the indicated name. The name appears in the table in 4-byte encoded form.
OpAnyElement	Matches an entire element with any name.
OpAnyElements	Matches any number of elements with any names.
OpAnyText	Matches a text token.
OpAttribute (name)	Matches the label token of an attribute with the indicated name. OpAttribute starts a clause that includes the subsequent clause in the table. The subsequent clause is used to match the value part of the attribute. OpAttribute clauses always appear after OpBeginElement or OpBeginAnyElement operations or after another OpAttribute clause.

Bytecode	Description
OpBeginChoice	OpBeginChoice starts a clause that ends with a matching OpEndChoice. Zero or more complete clauses must appear between OpBeginChoice and its matching OpEndChoice. The outer clause matches the tokens matched by any one of the inner clauses. All of the inner clauses must start with OpBeginElement except that the last one may be OpAnything. This construct corresponds to the XSD choice particle.
OpEndChoice	Ends a clause started with OpBeginChoice.
OpBeginSequence	OpBeginSequence starts a clause that ends with a matching OpEndSequence. Zero or more complete clauses must appear between OpBeginSequence and OpEndSequence. The outer clause matches the tokens matched by all the inner clauses in sequence. This construct corresponds to the XSD sequence particle.
OpEndSequence	Ends a clause started with OpBeginSequence.
OpBeginAll	OpBeginAll starts a clause that ends with a matching OpEndAll. Zero or more complete clauses must appear between OpBeginAll and OpEndAll. The outer clause matches the tokens matched by the inner clauses in any sequence. Occurrence operators relating to each inner clause indicate how many times the tokens for each clause may occur. The default is once. Multiple occurrences of a specified clause may be intermixed with occurrences of other inner clauses. All of the inner clauses must start with OpBeginElement except that the last one may be OpAnything. This construct corresponds to the XSD all particle.
OpEndAll	Ends a clause started with OpBeginAll.
OpAnything	Matches any number of elements and text tokens.
OpAnyNumber	Indicates that the subsequent clause may occur any number of times. OpAnyNumber starts a clause that terminates at the end of the subsequent clause.
OpOneOrMore	Indicates that the subsequent clause may occur one or more times. OpOneOrMore starts a clause that terminates at the end of the subsequent clause.
OpOptional	Indicates that the subsequent clause may occur zero or one times. OpOptional starts a clause that terminates at the end of the subsequent clause.
OpFormatInt8 (type,field)	Matches an 8-bit signed integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatInt8 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.

Bytecode	Description
OpFormatInt16 (type,field)	Matches a 16-bit signed integer. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatInt16 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatInt32 (type,field)	Matches a 32-bit signed integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatInt32 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatInt64 (type,field)	Matches a 64-bit signed integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatInt64 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatUInt8 (type,field)	Matches an 8-bit unsigned integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatUInt8 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatUInt16 (type,field)	Matches a 16-bit unsigned integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatUInt16 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatUInt32 (type,field)	Matches a 32-bit unsigned integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatUInt32 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatUInt64 (type,field)	Matches a 64-bit unsigned integer in a text token. A 4-byte offset argument indicates the offset of the binary representation in the current binary context. The OpFormatUInt64 macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatUnicodeString (type,field)	Matches a string consisting of the entire value of a text token. A 4-byte offset argument indicates the offset where a pointer to the string appears in the current binary context. The OpFormatUnicodeString macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.

Bytecode	Description
OpFormatDom (type,field)	Indicates that all XML matched by the subsequent clause is represented in binary form as DOM (a list of WSDXML_NODE structures). OpFormatDom starts a clause that terminates at the end of the subsequent clause. A 4-byte offset argument indicates the offset where a pointer to the first WSDXML_NODE appears in the current binary context. The OpFormatDom macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatStruct (structType,type,field)	Indicates that the binary context for the subsequent clause is a structure referenced by the current binary context. OpFormatStruct starts a clause that terminates at the end of the subsequent clause. The first of two 4-byte arguments indicates the size of the structure. The second argument indicates the offset where a pointer to the structure appears in the current binary context. The OpFormatStruct macro calculates the size of the indicated structure type (structType) and supplies the size as the first argument. The macro calculates the offset of the indicated field in the indicated type and supplies that offset as the second argument.
OpFormatUri (type,field)	Matches a URI in a text token. A 4-byte offset argument indicates the offset where a pointer to the URI string appears in the current binary context. The OpFormatUri macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatUuidUri (type,field)	Matches a UUID protocol URI in a text token. A 4-byte offset argument indicates the offset where URI in GUID structure form appears in the current binary context. The OpFormatUuidUri macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatName (type,field)	Matches a qualified name in a text token. A 4-byte offset argument indicates the offset where a pointer to the qualified name (a WSDXML_NAME structure) appears in the current binary context. The OpFormatName macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.

Bytecode	Description
OpFormatListInsertTail (nodeType,type,field)	Indicates that the binary contexts for all occurrences of the subsequent clause are structures in a singly-linked list referenced by the current binary context. OpFormatListInsertTail starts a clause that terminates at the end of the subsequent clause. The first of two 4-byte arguments indicates the size of the structures. The second argument indicates the offset where a pointer to the structure appears in the current binary context. The 'next' pointer that links the structures into a list is always the first field in the structures. The OpFormatListInsertTail macro calculates the size of the indicated structure type (nodeType) and supplies the size as the first argument. The macro calculates the offset of the indicated field in the indicated type and supplies that offset as the second argument.
OpFormatType (typetable,type,field)	Matches the tokens matched by the indicated type table and indicates that the binary context for that table is embedded in the current binary context at an indicated offset. The first of two 4-byte arguments is the encoded reference to the type table. The second argument is the offset. The OpFormatType macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatDynamicType (name,type,field)	Matches the tokens matched by a type table obtained dynamically using the indicated name and indicates that the binary context for that table is embedded in the current binary context at an indicated offset. The first of two 4-byte arguments is the name that identifies the type table. Typically, this name is specified in the form of a single-quoted string such as "'body'", which produces a 4-byte value. The second argument is the offset. The OpFormatDynamicType macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpFormatLookupType (urifield,type,field)	Matches the tokens matched by the type table identified by the URI at an indicated offset in the current binary context and indicates that the binary context for that table is embedded in the current binary context at an indicated offset. The first of two 4-byte arguments is the offset of the type URI in the current binary context. The second argument is the offset for the new binary context. The OpFormatLookupType macro calculates the offset of the indicated field in the indicated type and supplies that offset as the argument.
OpProcess(type,field)	Flags the referenced field as requiring additional processing during the generation and parsing phases. This is used for complex fields that cannot be automatically be processed.

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)