
Understanding Web Services Resource Access

Transfer, Eventing,
Enumeration and
MetadataExchange

Geoff Bullen, Dan Conti, Rick Hill

January 9, 2009

Authors

Geoff Bullen, Microsoft Corporation
Dan Conti, Microsoft Corporation
Rick Hill, Microsoft Corporation

Abstract

Understanding Web Resource Access is an introductory description of how to interact with web-based resources using Web Services. A number of examples are explored, with the intent to increase understanding and awareness of the various options available.

Copyright Notice

(c) 2008 Microsoft Corporation, Inc. All rights reserved

Microsoft (the "Author") hereby grants permission to copy and display the "Understanding Web Services Resource Access" paper (the "Document"), in any medium without fee or royalty, provided that you include the following on ALL copies of the Document that you make:

1. A link or URL to the Document at the Author's website.
2. The copyright notice as shown in the Document.

THE DOCUMENT IS PROVIDED "AS IS", AND THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE AUTHOR WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DOCUMENT.

The name and trademarks of the Author may NOT be used in any manner, including advertising or publicity pertaining to the Document or its contents without specific, written prior permission. Title to copyright in the Document will at all times remain with the Author.

No other rights are granted by implication, estoppel or otherwise.

Contents

1	Introduction	4
1.1	Disclaimer.....	4
1.2	Overview	4
1.3	Basic SOAP message contents	4
1.4	Web Services Specifications.....	5
2	Resource Access Overview – Let’s go Banking.....	6
2.1	Woodgrove Bank defines a bank account	7
2.2	Getting Todd’s bank account resource	7
2.3	How can Todd find out about that transaction history request?	8
2.4	Todd gets too much information back.....	10
2.4.1	Start enumerating over the transaction information	11
2.4.2	Retrieve the transaction items in pieces.....	12
2.5	Getting an event when the bank account changes	13
2.5.1	Subscribing to bank account events	14
2.5.2	Receiving the account change events.....	15
2.5.3	Filtering the number of events Todd receives	15
3	WS-Transfer	16
3.1	Transfer Create	16
3.2	Transfer Put.....	17
3.3	Transfer Delete	18
4	WS-MetadataExchange.....	18
4.1	Using GetMetadata.....	18
4.2	Using metadata in an EPR	19
5	WS-Enumeration.....	20
5.1	Filtering	20
6	WS-Eventing.....	20
6.1	Using a Subscription Manager	20
6.2	Using Subscription End	21
6.3	Subscription Identification.....	23
7	Appendix A: Acknowledgements	23
8	Appendix B: XML Namespaces.....	24

1 Introduction

This section provides relevant background information associated with the Resource Access collection of specifications and the white paper itself.

1.1 Disclaimer

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

1.2 Overview

The primary goal of this white paper is to increase understanding of the WS-Transfer [**WS-Transfer**], WS-Eventing [**WS-Eventing**], WS-Enumeration [**WS-Enumeration**] and WS-MetadataExchange [**WS-MEX**] specifications (collectively known as the “Resource Access” specifications) by introducing appropriate usage scenarios and later discussing important specification details. Readers should refer to the respective specifications for complete normative description and semantics of the concerned protocols. WS-ResourceTransfer [**WS-RT**] is beyond the current scope of this document.

This document is for:

- Web architects looking to understand how these WS-* standards can be used to build service-oriented application architectures and use a common resourcing framework. It will also help explain how best to compose these standards with other WS-* standards.
- Web Service implementers looking to build specific application components that use these specifications.

This document assumes a general understanding of XML, Namespaces in XML, WS-Addressing, WSDL and SOAP.

1.3 Basic SOAP message contents

Example 1-3-1 shows a SOAP [**SOAP**] message that is requesting the transaction history for a particular account. It obviously ignores security, scalability, reliability, and other considerations, all of which would need to be in place for a real service of this type to execute correctly. It also generally ignores namespace prefix definitions (which can be found in Appendix B) and addressing elements such wsa:MessageID in order to make the example smaller. Please see the WS-Addressing specification [**WS-Addressing**] for complete details of the required and optional addressing elements.

Example 1-3-1. SOAP Request Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/TransactionHistory</wsa:To>
    <wsa:ReplyTo>
```

```

    <wsa:Address>http://todd.adatum.com/customer</wsa:Address>
  </wsa:ReplyTo>
  <wsa:Action>http://x.woodgrovebank.com/GetHistory</wsa:Action>
  <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
  ...
</soap:Header>
<soap:Body />
</soap:Envelope>

```

Note in particular `wsa:Action` which describes the type of action to be performed by the TransactionHistory service and also the user-defined reference parameter `bank:AccID` which, in this case, is used to identify the account involved in this operation.

The TransactionHistory service would respond with a message similar to that found in Example 1-3-2 below. Again, much of the associated addressing information has been omitted for brevity.

Example 1-3-2. SOAP Response Message

```

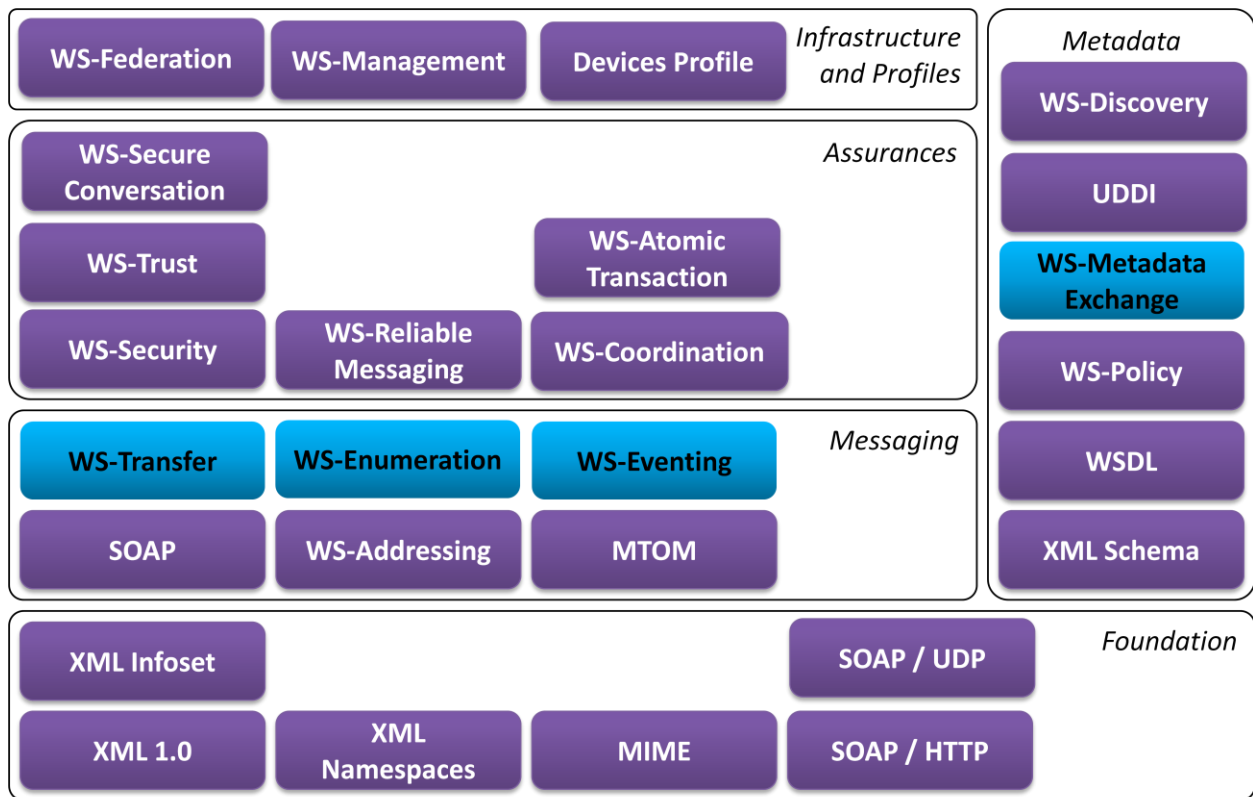
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://todd.adatum.com/customer</wsa:To>
    <wsa:Action>http://x.woodgrovebank.com/HistoryResponse</wsa:Action>
    <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
  </soap:Header>
  <soap:Body>
    <bank:Transactions>
      <bank:Transaction id="t14324"> info </bank:Transaction>
      <bank:Transaction id="t14325"> info </bank:Transaction>
      <bank:Transaction id="t14326"> info </bank:Transaction>
      ...
    </bank:Transactions>
  </soap:Body>
</soap:Envelope>

```

Almost all of the examples that follow will be building on this basic request/response message pattern. WS-Eventing is an asynchronous protocol and therefore those examples will follow a slightly different pattern. In all cases, only elements that are required to add clarity will be shown.

1.4 Web Services Specifications

The following diagram outlines the major Web Services specifications. The four highlighted specifications are the ones specifically used in this white paper. Please note that WS-Transfer, WS-Eventing, WS-Enumeration and WS-MetadataExchange can be composed with other specifications.



2 Resource Access Overview – Let’s go Banking

The Resource Access specifications provide a number of standard interfaces that, when used together, can help solve some of the basic issues associated with accessing remote information and resources. Throughout this example we will be thinking about accessing information about a bank account. The basic scenario is that Todd Meadows wants to access his Woodgrove bank account. Woodgrove Bank provides Resource Access Web Services which Todd can use to accomplish this task. Todd is a programmer at heart and does not believe in any of this new fangled GUI stuff – “anything worth doing is worth doing with Web Services” is his motto.

Firstly Todd wants to see how much money he has in his bank account (WS-Transfer). When Todd sees his balance, he doesn’t think he has the right amount of money available, and so wants to get his transaction history to find out what is going on. Unfortunately he does not exactly know how to execute this operation and must first understand what is required in order to successfully see this information (WS-MetadataExchange). Having successfully executed the operation to get his transaction history, Todd now realizes that the number of records is far too long to be retrieved in one call, and so he wants to break this down into a number of more manageable pieces of information (WS-Enumeration). Finally Todd, after searching through his newly-chunked account transactions, has found out that some strange person has been accessing his account without permission, and so now he wants to be notified whenever any information in his account changes (WS-Eventing).

This example, while simplified, will demonstrate some important aspects of each Resource Access protocol as well as providing insight into useful usage patterns. Later sections will cover more advanced features.

2.1 Woodgrove Bank defines a bank account

A resource can be anything that is addressable via an Endpoint Reference (EPR), as defined in WS-Addressing [WS-Addressing], and is able to be represented in XML format, for example: a customer, an order, a bank account, etc. This representation of a resource is normally left up to the application designers (in this case the bank) to decide upon, or potentially for other standards bodies to work out.

Todd talks to Woodgrove Bank and finds out they have defined their bank account resource to be of the following form:

Example 2-1-1. Bank Account Resource

```
<bank:Account>
  <bank:AccID>a1234567</bank:AccID>
  <bank:First>Todd</bank:First>
  <bank:Last>Meadows</bank:Last>
  <bank:SSN>123456789</bank:SSN>
  <bank:Balance>10000</bank:Balance>
</bank:Account>
```

Todd also needs to know how to directly address a bank account resource using an EPR. Woodgrove Bank tells him to use the following EPR to access his account:

Example 2-1-2. Bank Endpoint Reference

```
<wsa:EndPointReference>
  <wsa:Address>http://x.woodgrovebank.com/account</wsa:Address>
  <wsa:ReferenceParameters>
    <bank:AccID>a1234567</bank:AccID>
  </wsa:ReferenceParameters>
</wsa:EndPointReference >
```

Note that the bank uses an EPR that contains a reference parameter (`bank:AccID`) to uniquely define the bank account to use. Other implementations may choose different models, for example: embedding the account information in the URI itself (`http://x.woodgrovebank.com/account/a1234567`).

2.2 Getting Todd's bank account resource

There are a common, basic set of operations that need to be performed on any resource. These are: create, read, update and delete – the so-called “CRUD” operations. The WS-Transfer specification provides this functionality through the following actions: Transfer **Create**, Transfer **Get**, Transfer **Put** and Transfer **Delete**.

Todd wants to **get** his current bank account information from Woodgrove Bank and so will use the Transfer **Get** message. The bank has already provided him with the EPR to use to access his account.

Example 2-2-1. Get Message

```
<soap:Envelope>
```

```

<soap:Header>
  <wsa:To>http://x.woodgrovebank.com/account</wsa:To>
  <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
  <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
  </wsa:Action>
</soap:Header>
<soap:Body />
</soap:Envelope>

```

Example 2-2-2. Get Response Message

```

<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
    </wsa:Action>
    <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
  </soap:Header>
  <soap:Body>
    <bank:Account>
      <bank:AccID>a1234567</bank:AccID>
      <bank:First>Todd</bank:First>
      <bank:Last>Meadows</bank:Last>
      <bank:SSN>123456789</bank:SSN>
      <bank:Balance>10000</bank:Balance>
    </bank:Account>
  </soap:Body>
</soap:Envelope>

```

Notes:

- The body of the **Get Response** contains the XML representation of the resource, which in this case is the bank account resource described in Example 2-1-1.

2.3 How can Todd find out about that transaction history request?

One special type of resource, defined by WS-MetadataExchange, is called a metadata resource. WS-MetadataExchange provides a format for representing metadata about a Web Service. This metadata can include XML information, such as WSDL and policy assertions, which can be used to determine how to successfully access and use a Web Service.

In this case Todd’s problem is that he does not know how to correctly access the “TransactionHistory” Web Service that is used to obtain his transaction history. Todd can use WS-MetadataExchange to solve this problem. The metadata returned using WS-MetadataExchange has a specific form, which allows Todd to interpret the contents and understand a lot of information about how to access the “TransactionHistory” service. An example of a metadata resource can be seen below:

Example 2-3-1. Metadata Resource Format

```

<mex:Metadata>
  <mex:MetadataSection Dialect='http://schemas.xmlsoap.org/wsdl/'>
    <wsdl:definitions ... >
      ...
    </wsdl:definitions>

```

```

</mex:MetadataSection>
<mex:MetadataSection Dialect='http://www.w3.org/2001/XMLSchema' ... >
...
</mex:MetadataSection>
<mex:MetadataSection
  Dialect='http://schemas.xmlsoap.org/ws/2004/09/policy' ... >
  <mex:MetadataReference>
    <wsa:Address>
      http://x.woodgrovebank.com/TransactionHistory/policy
    </wsa:Address>
  </mex:MetadataReference>
</mex:MetadataSection>
</mex:Metadata>

```

Notes:

- Web Services metadata is a collection of metadata sections, each defined by a separate `mex:MetadataSection` element as seen above. Each section represents a particular type (or Dialect) of metadata. For example, the section above defined by `Dialect='http://schemas.xmlsoap.org/wsd1/'` would contain WSDL metadata about the TransactionHistory service.
- The WS-MetadataExchange specification defines a number of dialects, including XML Schema, WSDL, Policy, Policy Attachment and Metadata (MEX itself). Other dialects can be defined by other specifications or the application itself.
- It is also possible to return metadata as part of an EPR (by embedding the `wsa:Metadata` element, which is part of the WS-Addressing specification), rather than having to fetch the information separately. More information about this option is available in the WS-MetadataExchange [WS-MEX] specification.
- The `mex:MetadataReference` element can be used to provide an endpoint reference to a metadata resource, for example policy documents (as seen in the example above).

Here is how Todd went about retrieving this metadata information from Woodgrove Bank.

Example 2-3-2. Metadata Request Message

```

<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/TransactionHistory/mex</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa:Action>
  </soap:Header>
  <soap:Body />
</soap:Envelope>

```

Example 2-3-3. Metadata Response Message

```

<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
    </wsa:Action>
  </soap:Header>

```

```
<soap:Body>
  <mex:Metadata>
    ... metadata resource information here - see Example 2-3-1
  </mex:Metadata>
</soap:Body>
</soap:Envelope>
```

Notes:

- Metadata can be retrieved using the Transfer **Get** action. Every metadata resource is always a Transfer resource and therefore it implements the **Get** operation.
- Because metadata operations occur far less frequently than resource operations, the bank implemented a separate metadata service (<http://x.woodgrovebank.com/TransactionHistory/mex>) which returns metadata about the TransactionHistory Web service. This separation is recommended to allow resource operations to be scaled appropriately.

2.4 Todd gets too much information back

Sometimes the size of the resource representation (or dataset) being accessed can be very large. WS-Enumeration can be used in these cases to allow subsets of the data to be returned with each subsequent call, until the data is exhausted.

In this case, Todd realizes that there is too much information returned from the TransactionHistory Web Service (see Example 1-3-1) and he wants to find a way to retrieve the history information in smaller chunks.

From the WSDL [**WSDL**] associated with TransactionHistory, which was retrieved as part of the metadata (see Example 2-3-3), Todd finds that the service supports WS-Enumeration and decides to use it.

The basic flow Todd uses is as follows:

1. Send an **Enumerate** message to TransactionHistory asking to enumerate it.
2. An opaque **Enumeration Context** (implementation specific XML data) is returned.
3. Enumerate over the transaction data using a **Pull** message.
4. Each **Pull** message will result in a set of transaction items being returned.
5. Continue 3 and 4 until no more data is available or until Todd finds what he is looking for.

When Todd sends the **Enumerate** message, it will return an opaque **Enumeration Context** of the form below. This context must be passed in every subsequent **Pull** message.

Example 2-4-1. Enumeration Context

```
<wsen:EnumerationContext>...</wsen:EnumerationContext>
```

Each **Pull** message Todd sends will return a set of transaction items of the form:

Example 2-4-2. Enumeration Items

```
<wsen:Items>
```

```

<bank:Transaction id="t14324"> info </bank:Transaction>
<bank:Transaction id="t14325"> info </bank:Transaction>
<bank:Transaction id="t14326"> info </bank:Transaction>
...
</wsen:Items>

```

In the bank scenario, the service Todd used to get his bank account transaction history (TransactionHistory) also supports WS-Enumeration to handle larger datasets. Let us look at what Todd does to use this.

2.4.1 Start enumerating over the transaction information

First Todd sends a message to “TransactionHistory” to start the enumeration. He will get back an **Enumeration Context** to be used later.

Example 2-4-3. Enumerate Message

```

<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/TransactionHistory</wsa:To>
    <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wsen:Enumerate>
    </wsen:Enumerate>
  </soap:Body>
</soap:Envelope>

```

Example 2-4-4. Enumerate Response Message

```

<soap:Envelope>
  <soap:Header>
    <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wsen:EnumerateResponse>
      <wsen:EnumerationContext>E#12345</wsen:EnumerationContext>
    </wsen:EnumerateResponse>
  </soap:Body>
</soap:Envelope>

```

Notes:

- wsen:EnumerationContext will always be returned and will contain opaque data that uniquely represents this enumeration. This will be used in subsequent **Pull** messages.
- There are a number of optional elements Todd can pass in the initial **Enumerate** message. More details on these and other elements can be found in the WS-Enumeration specification [WS-Enum]. These options include:

- **Expires** – Requests that the Enumeration expire after a certain interval and no longer be valid. By default the **Enumeration Context** will not expire.
- **Filter** – This allows you to specify a filter on the set of items you want returned. Only items that match the filter will be returned in subsequent **Pull** messages. By default all items are returned
- There are a number of optional elements returned in an **Enumerate Response** message. More details on these and other elements can be found in the WS-Enumeration specification [WS-Enum]. These options include:
 - **Expires** – The expiration time actually given to the enumeration.
- The TransactionHistory service supports retrieving all transaction history items at once (see Examples 1-3-2 and 1-3-2), but it can also support retrieving the same information in smaller chunks via WS-Enumeration. Todd can examine the resource metadata associated with a Web Service in order to determine whether or not a Web Service supports WS-Enumeration.

2.4.2 Retrieve the transaction items in pieces

Now that Todd has successfully started an enumeration, he can send a number of **Pull** messages to retrieve the transaction information in chunks from the bank. Let's look at how Todd does this.

Example 2-4-5. Pull Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/TransactionHistory</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wsen:Pull>
      <wsen:EnumerationContext>E#12345</wsen:EnumerationContext>
    </wsen:Pull>
  </soap:Body>
</soap:Envelope>
```

Example 2-4-6. Pull Response Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wsen:PullResponse>
      <wsen:Items>
        <bank:Transaction id="t14324"> info </bank:Transaction>
        <bank:Transaction id="t14325"> info </bank:Transaction>
        <bank:Transaction id="t14326"> info </bank:Transaction>
        ...
      </wsen:Items>
      <wsen:EndOfSequence />
    </wsen:PullResponse>
  </soap:Body>
```

```
</soap:Envelope>
```

Notes:

- Every **Pull** message must contain an **Enumeration Context**. The Pull Response message may optionally contain a new **Enumeration Context**, which must then be used in all subsequent Pull messages.
- A **Pull Response** message will normally contain a set of Items.
- A **Pull Response** message may contain an **EndOfSequence** element, which indicates that there are no more items to be returned. In fact it also indicates that the **Enumeration Context** is no longer valid and has been deleted. Once this element has been received, do not try to send any more **Pull** messages, as they will cause faults. It would not normally be useful to return both a new **Enumeration Context** and an **EndOfSequence** element in the same message.
- Todd may find the particular transaction he is looking for in one of the early **Pull Response** messages he received, and may no longer want to issue further (redundant) **Pull** messages to retrieve the remainder of the items available. In this case Todd could explicitly delete the **Enumeration Context** by sending a **Release** message.
- If more time is needed to process the enumeration than expected, use the **Renew** message to update the expiration time for the enumeration.

2.5 Getting an event when the bank account changes

When accessing resources and indeed in using Web Services in general there are a number of scenarios where asynchronous messaging is required. For example:

- If a task takes a long time to complete
- If a system generates alerts or events at some future time that need to be delivered to a variety of destinations

WS-Eventing provides a mechanism to allow clients (event sinks) to **Subscribe** to messages (notifications) generated by a server (event source). An event sink will then receive message notifications from an event source, as needed. WS-Eventing also provides other functionality, including a way for an event sink to **Unsubscribe** to a set of messages.

In the banking scenario, by looking through the transaction information, Todd realizes that some strange person has been taking money from his account without his permission. In order to be more quickly alerted to future occurrences of this problem, Todd wants to be notified whenever any details about his account change. Todd accomplishes this by using WS-Eventing.

In WS-Eventing terms, Todd is an event sink and his bank is an event source. First, Todd will need to **Subscribe** to notifications from the bank. Then, the bank will send him notification messages whenever anything changes.

2.5.1 Subscribing to bank account events

In this first attempt, Todd **Subscribes** to all notifications about accounts that the bank has to offer. Todd will realize later that there will be far too many such events and he will have to put a filter on the subscription to only receive events concerning his particular bank account.

Here is the **Subscribe** message that Todd initially uses:

Example 2-5-1. Subscribe Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/account</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wse:Subscribe>
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://todd.adata.com/customer</wsa:Address>
        </wse:NotifyTo>
      </wse:Delivery>
    </wse:Subscribe>
  </soap:Body>
</soap:Envelope>
```

Notes:

- The `wse:NotifyTo` element defines the EPR where notification messages will be sent (event sink). Normally this EPR refers back to the client process that is sending the **Subscribe** message (as it does above), but it could also refer to a separate process that handles the events on behalf of the client.
- There are a number of option elements you can set in the **Subscribe** message. These include:
 - **Expires** – This allows the client to request a time limit on the subscription.
 - **Filter** – This allows the client to specify a filter on the set of notification messages being sent.

Here is the response message that Todd receives from his **Subscribe** message:

Example 2-5-2. Subscribe Response Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wse:SubscribeResponse>
      <wse:SubscriptionManager>
        <wsa:Address>http://x.woodgrovebank.com/account</wsa:Address>
        <wsa:ReferenceParameters>
```

```

    <wse:Identifier>uuid:22e8a584-0d18-4228-b2a8</wse:Identifier>
  </wsa:ReferenceParameters>
</wse:SubscriptionManager>
  <wse:Expires>2010-07-01T00:00:00.000-00:00</wse:Expires>
</wse:SubscribeResponse>
</soap:Body>
</soap:Envelope>

```

Notes:

- The `wse:SubscriptionManager` element defines the EPR where all future messages concerning this subscription should be sent (for example: **Unsubscribe**, **Renew**, etc). Normally the subscription manager will be the same as the event source, but this need not be the case.
- Note the use of reference parameters (`wsa:ReferenceParameters` element) to uniquely define this subscription. In this case the event source has chosen to use the optional `wse:Identifier` element to provide this unique subscription identification.
- The `wse:Expires` element returns the date/time (if any) when the subscription will finish. It is possible to extend the duration of a subscription by using a **Renew** message before expiration.

2.5.2 Receiving the account change events

Now that Todd has successfully subscribed to all event notifications from the bank, he will start to receive events from the bank of the form below:

Example 2-5-3. Event Notification

```

<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://x.woodgrovebank.com/notifications/AccountChange
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <bank:AccountChange>
      ...
    </bank:AccountChange>
  </soap:Body>
</soap:Envelope>

```

Notes:

- Both the `wsa:Action` and the `soap:Body` contents are defined by the event source.

2.5.3 Filtering the number of events Todd receives

Todd quickly sees that he will be overrun with event notifications, as he starts to receive thousands of events which represent the account changes occurring on everyone's bank accounts (clearly this would never happen in real life). He now realizes that he needs to use a filter as part of his **Subscribe** request so that he only receives events from his specific account. To accomplish this Todd **Unsubscribes** to (or terminates) the current subscription and then creates a new subscription, this time including the filter element below:

Example 2-5-4. Filtering Events

```
<wse:Subscribe>
  ...
  <wse:Filter Dialect="http://x.woodgrovebank.com/accountFilter">
    a1234567
  </wse:Filter>
</wse:/Subscribe>
```

Notes:

- WS-Eventing defines the XPath [**XPath**] filter dialect, and also provides extension points to allow for the creation of application-specific filtering mechanisms, as shown above. In this case the bank provides a very simple filter to specify the account of interest.

3 WS-Transfer

In the banking scenario we discussed the Transfer **Get** verb. In this section we cover the other verbs defined in the WS-Transfer specification.

3.1 Transfer Create

WS-Transfer supports the concept of creating a new resource. The following example highlights some of the fundamentals of the Transfer **Create** and **Create Response** messages.

Example 3-1-1. Transfer Create Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/accountFactory</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <bank:Account>
      <bank:First>Todd</bank:First>
      <bank:Last>Meadows</bank:Last>
      <bank:SSN>123456789</bank:SSN>
    </bank:Account>
  </soap:Body>
</soap:Envelope>
```

Example 3-1-2. Transfer Create Response Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wxf:ResourceCreated>
      <wsa:Address>http://x.woodgrovebank.com/account</wsa:Address>
      <wsa:ReferenceParameters>
        <bank:AccID>a1234567</bank:AccID>
      </wsa:ReferenceParameters>
    </wxf:ResourceCreated>
  </soap:Body>
</soap:Envelope>
```

```

    </wsa:ReferenceParameters>
  </wxf:ResourceCreated>
</soap:Body>
</soap:Envelope>

```

Notes:

- Please note that Transfer **Create** requests are sent to a different interface than Transfer **Get, Put** and **Delete** requests, according to the WS-Transfer spec. Transfer **Create** requests go to the resource factory interface, which in this example is called “accountFactory.”
- While you normally send the request to a generic EPR used to create resources (accountFactory), the body of the response (inside the `wxf:ResourceCreated` element) can specify a different EPR where future Transfer **Get, Put** and **Delete** messages for this specific resource should be sent (account).
- A `wxf:ResourceCreated` element will always be returned if the resource was successfully created.

3.2 Transfer Put

WS-Transfer supports the concept of updating the information in a specific resource. To do this you must pass the entire contents of the updated resource. This action does not attempt to handle the problems associated with simultaneous updates, which can be dealt with through composition with appropriate Web Services standards.

Example 3-2-1. Transfer Put Message

```

<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/account</wsa:To>
    <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <bank:Account>
      <bank:AccID>a1234567</bank:AccID>
      <bank:First>Todd</bank:First>
      <bank>Last>Meadows</bank>Last>
      <bank:SSN>123456789</bank:SSN>
    </bank:Account>
  </soap:Body>
</soap:Envelope>

```

Example 3-2-2. Transfer Put Response Message

```

<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body/>
</soap:Envelope>

```

Notes:

- The response will only return the contents of the resource in the body if it differs from that which was sent in the Transfer **Put** message.

3.3 Transfer Delete

WS-Transfer also supports the concept of deleting a specific resource.

Example 3-3-1. Transfer Delete Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/account</wsa:To>
    <bank:AccID wsa:IsReferenceParameter="true">a1234567</bank:AccID>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
    </wsa:Action>
  </soap:Header>
  <soap:Body/>
</soap:Envelope>
```

Example 3-3-2. Transfer Delete Response Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body/>
</soap:Envelope>
```

Notes:

- A delete response message contains an empty SOAP body.

4 WS-MetadataExchange

This section covers some of the more advanced features of the WS-MetadataExchange specification.

4.1 Using GetMetadata

WS-MetadataExchange also defines how a Web service can optionally support a GetMetadata operation that can be used to retrieve service metadata. The example below shows how Todd might have used **GetMetadata** to return the metadata associated with the TransactionHistory Web Service.

Example 4-1-1. GetMetadata Request Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://x.woodgrovebank.com/TransactionHistory</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Request
    </wsa:Action>
  </soap:Header>
```

```
<soap:Body />
</soap:Envelope>
```

Example 4-1-2. GetMetadata Response Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Response
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <mex:Metadata>
      ... metadata resource information here - see Example 2-3-1
    </mex:Metadata>
  </soap:Body>
</soap:Envelope>
```

Notes:

- Support by a Web Service for the **GetMetadata** request is optional and cannot be assumed.
- Note that in the example above the request is sent directly to the TransactionHistory Web Service while in Example 2-3-2 the Transfer **Get** request is sent to another endpoint (<http://x.woodgrovebank.com/TransactionHistory/mex>) which returns the metadata about TransactionHistory.

Some useful implementation principles to consider when deciding on the correct implementation strategy for returning metadata are given below:

- Metadata retrieval happens relatively infrequently compared to the frequency of operations on Web Services, so implementers often separate metadata retrieval from other real time retrieval.
 - It is important to optimize real time operations over metadata retrieval.
- Often metadata retrieval has weaker (less secure) bindings or different bindings than real time operations. For instance, getting the metadata for a GetSecurityToken service could have weaker security requirements than actually getting a security token. This could lead to an implementation decision to separate the metadata resource role from a service.

4.2 Using metadata in an EPR

WS-MetadataExchange provides a mechanism to embed resource metadata inside the EPR of the Web Service itself. This can help in bootstrapping an interaction and can also save the extra round trip of requesting the metadata via a Transfer **Get** once you have the EPR. However, embedding the metadata in the EPR may cause some performance problems if the returned metadata is large. WS-Addressing allows metadata to be added directly to an EPR in the following way:

Example 4-2-1. Metadata in an EPR

```
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:Metadata>
    <mex:Metadata>
      <mex:MetadataSection Dialect='http://schemas.xmlsoap.org/wsdl/'>
```

```

        <wsdl:definitions ... >
        ...
        </wsdl:definitions>
    </mex:MetadataSection>
    ...
    </mex:Metadata>
    ...
    </wsa:Metadata>
    ...
</wsa:EndpointReference>

```

5 WS-Enumeration

This section covers some of the more advanced features of the WS-Enumeration specification.

5.1 Filtering

In the same way that event notifications were filtered in the earlier bank example (Example 2-5-4) so that only the desired subset of events were sent to the event sink, WS-Enumeration allows you to filter the dataset to be returned via the **Pull** request. It works in a very similar way to the filtering in WS-Eventing.

Example 5-1-1. Enumeration Filter

```

<wsen:Enumerate>
...
  <wsen:Filter Dialect="http://x.woodgrovebank.com/enumFilter">
    ...
  </wsen:Filter>
</wsen:/Enumerate>

```

Notes:

- WS-Enumeration defines the XPath **[XPath]** filter dialect, and also provides extension points to allow for the creation of application-specific filtering mechanisms, as shown above.

6 WS-Eventing

This section covers some of the more advanced features of the WS-Eventing specification.

6.1 Using a Subscription Manager

A separate subscription manager is used in cases where the event environment being implemented needs to be more flexible and scalable. In the simplest case, the subscription manager is the same as the event source. This is the case shown in the bank scenario (Example 2-5-2). The response to the **Subscribe** message was:

Example 6-1-1. Subscribe Response

```

<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
    </wsa:Action>
  </soap:Header>
</soap:Envelope>

```

```

    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wse:SubscribeResponse>
      <wse:SubscriptionManager>
        <wsa:Address>http://x.woodgrovebank.com/account</wsa:Address>
        <wsa:ReferenceParameters>
          <wse:Identifier>uuid:22e8a584-0d18-4228-b2a8</wse:Identifier>
        </wsa:ReferenceParameters>
      </wse:SubscriptionManager>
      <wse:Expires>2010-07-01T00:00:00.000-00:00</wse:Expires>
    </wse:SubscribeResponse>
  </soap:Body>
</soap:Envelope>

```

It included the EPR of the subscription manager which contained a URI which was the same as the event source itself. But it is also possible for the EPR returned in the **Subscribe Response** to be different. For example:

Example 6-1-2. Subscribe Response using a Subscription Manager

```

<soap:Envelope>
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
    </wsa:Action>
  </soap:Header>
  <soap:Body>
    <wse:SubscribeResponse>
      <wse:SubscriptionManager>
        <wsa:Address>http://x.woodgrovebank.com/account_manager</wsa:Address>
        <wsa:ReferenceParameters>
          <wse:Identifier>uuid:22e8a584-0d18-4228-b2a8</wse:Identifier>
        </wsa:ReferenceParameters>
      </wse:SubscriptionManager>
      <wse:Expires>2010-07-01T00:00:00.000-00:00</wse:Expires>
    </wse:SubscribeResponse>
  </soap:Body>
</soap:Envelope>

```

Here a new URI “account_manager” is being returned. All subsequent requests, such as **Renew**, **GetStatus** and, importantly, **Unsubscribe** will now need to be sent to “account_manager”. The event source (in this case http://x.woodgrovebank.com/account) is still responsible for sending out events. It is important to note that it is up to the implementation to provide whatever mechanisms are required in order to keep the event source and the event manager in sync.

6.2 Using Subscription End

It is often important for an event sink to be informed when a subscription that the event sink created has ended for some reason. As part of a **Subscribe** message sent by an event sink to an event source, an event sink can send an “EndTo” EPR as follows:

Example 6-2-1. Subscribe Message using EndTo

```

<soap:Envelope>

```

```

<soap:Header>
  <wsa:To>http://x.woodgrovebank.com/account</wsa:To>
  <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
  </wsa:Action>
</soap:Header>
<soap:Body>
  <wse:Subscribe>
    <wse:Delivery>
      <wse:NotifyTo>
        <wsa:Address>http://todd.adatum.com/customer</wsa:Address>
      </wse:NotifyTo>
    </wse:Delivery>
    <wse:EndTo>
      <wsa:Address>http://todd.adatum.com/customer</wsa:Address>
    </wse:EndTo>
  </wse:Subscribe>
</soap:Body>
</soap:Envelope>

```

In this case the EPR of the `wse:EndTo` is the same as the event sink EPR. This does not have to be the case, however. It is important to note that the process associated with an `wse:EndTo` EPR needs to be able to accept and process **SubscriptionEnd** messages that are sent to it. These messages are of the form:

Example 6-2-2. SubscriptionEnd Message

```

<soap:Envelope ...>
  <soap:Header ...>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd
    </wsa:Action>
  </soap:Header>
  <soap:Body ...>
    <wse:SubscriptionEnd>
      <wse:SubscriptionManager>
        <wsa:Address>http://x.woodgrovebank.com/account_manager</wsa:Address>
      </wse:SubscriptionManager>
      <wse:Status>
        http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown
      </wse:Status>
      <wse:Reason>System reboot required</wse:Reason>
    </wse:SubscriptionEnd>
  </soap:Body>
</soap:Envelope>

```

Notes:

- The subscription manager EPR is returned as part of the message.
- An Eventing-defined status `<wse:Status>` is returned which can be matched to determine why the subscription has ended.
- An implementation dependent string `<wse:Reason>` is also returned to provide further explanation of the situation.

6.3 Subscription Identification

It is very common for one endpoint to handle multiple subscription notifications or for one subscription manager to handle multiple types of subscriptions. In order to achieve this, implementations must be aware of how to handle subscription identification.

Every subscription must be uniquely identified by an event source via an EPR (returned in the **Subscribe Response**), so that subsequent operations (**Renew**, **Unsubscribe**, etc) can be performed on the appropriate subscription. While it is up to the implementer of the event source to provide the means to ensure that the identification is unique, there is a simple method that, while optional, is recommended in normal circumstances. WS-Eventing defines a reference parameter called `wse:Identifier` which contains an opaque GUID to uniquely identify a subscription. While the GUID itself (normally a string of some sort) is created in an implementation-dependent manner, the fact that the subscriber can generically distinguish between subscription manager EPRs by comparing the contents of `wse:Identifier` can be helpful in event sink implementations.

It is useful to now look at the flow of identification that typically occurs during the lifecycle of a subscription. Subscription identification is typically implemented using reference parameters as is shown in the following example:

A **Subscribe** message is sent to the event source that contains:

1. A user-defined reference parameter A in the `NotifyTo` EPR in the body
2. A user-defined reference parameter B in the `EndTo` EPR in the body

The event source responds with a **Subscribe Response** message that may contain:

1. Event Source-defined `wse:Identifier` C in the subscription manager EPR in the body

The Event source will send event notification messages that contain:

1. User-defined reference parameter A in the header

The Event source will send a **Subscription End** message that contains:

1. User-defined reference parameter B in the header
2. Event Source-defined `wse:Identifier` C in the body

7 Appendix A: Acknowledgements

The following people provided very valuable help and assistance in developing this white paper: Asir Vedamuthu, Paul Cotton, Greg Carpenter, Dan Driscoll, Colleen Evans, Doug Bunting, Monica Martin, Ram Jeyaraman, Mark Wahl, Joe Schulman, Toby Nixon and Geoffrey Creighton.

8 Appendix B: XML Namespaces

The table below lists XML Namespaces that are used in this document. The choice of any namespace prefix is arbitrary and not semantically significant.

Table B-1. Prefixes and XML Namespaces used in this specification.

Prefix	XML Namespace	Specification
soap	http://www.w3.org/2003/05/soap-envelope	[SOAP 1.2]
wsa	http://www.w3.org/2005/08/addressing	[WS-Addressing]
wsdl	http://schemas.xmlsoap.org/wsdl/	[WSDL 1.1]
wxf	http://schemas.xmlsoap.org/ws/2004/09/transfer	[WS-Transfer]
wse	http://schemas.xmlsoap.org/ws/2004/08/eventing	[WS-Eventing]
wsen	http://schemas.xmlsoap.org/ws/2004/09/enumeration	[WS-Enumeration]
mex	http://schemas.xmlsoap.org/ws/2004/09/mex	[WS-MetadataExchange]
bank	http://schemas.woodgrovebank.com/bank	Owned by the bank

9 Appendix C: References

[[WS-MEX](#)] Web Services Metadata Exchange (WS-MetadataExchange)
<http://www.w3.org/Submission/WS-MetadataExchange/>

[[WS-Transfer](#)] Web Services Transfer (WS-Transfer)
<http://www.w3.org/Submission/WS-Transfer/>

[[WS-Eventing](#)] Web Services Eventing (WS-Eventing)
<http://www.w3.org/Submission/WS-Eventing/>

[[WS-Enumeration](#)] Web Services Enumeration (WS-Enumeration)
<http://www.w3.org/Submission/WS-Enumeration/>

[[WS-RT](#)] Web Services Resource Transfer (WS-RT)
<http://www.w3.org/Submission/WSRT/>

[[SOAP](#)] Simple Object Access Protocol (SOAP) 1.2
<http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>

[[WS-Addressing](#)] Web Services Addressing (WS-Addressing)
<http://www.w3.org/2005/08/addressing>

[[WSDL](#)] Web Services Description Language (WSDL)
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

[[XPath](#)] XML Path Language Version 1.0

<http://www.w3.org/TR/1999/REC-xpath-19991116>