



Technology Dedicated to Business Efficiency

Web Services

Web Services v4.0s002

Web Services: Web Services v4.0s002

Copyright © 2009 Standards for Technology in Automotive Retail

Editor:

David Carver, STAR

Jason Loeffler, Karmak

Contributors:

Ramesh Rangaiah, Navistar

Pejavar Rao, Navistar

Russell Shephard, T-Systems

Table of Contents

I. Preface	xiii
I.I. Purpose	xiii
I.II. SCOPE	xiii
I.III. AUDIENCE	xiv
I.IV. BACKGROUND	xiv
I.V. SERVICE PROVIDER REQUIREMENTS	xiv
I.VI. COMMUNICATION PATTERNS OVERVIEW	xv
Part I. STAR Level One	1
1. STAR Web Services Overview	3
1.1. Background	3
1.2. STAR Web Services Types	3
1.3. Web Service Interoperability Requirements	3
2. Common Components	5
2.1. Overview	5
2.2. Message Packaging	5
2.2.1. Notes Regarding Payloads and Attachments	7
2.3. Namespaces	7
2.4. Web Methods	8
2.4.1. ProcessMessage	8
2.4.2. PutMessage	11
2.4.3. PullMessage	12
2.5. The payload Manifest SOAP Header	14
3. Communication Patterns	17
3.1. One-Way Communication	17
3.1.1. One-Way Synchronous Communication	17
3.1.2. One-Way Asynchronous Communication	17
3.2. Two-Way Communication	18
3.2.1. Two-Way Synchronous Communication	18
3.2.2. Two-Way Asynchronous Communication	18
4. Generic Web Services Specifications	21
4.1. Overview	21
4.2. Generic WSDL	21
4.3. Benefits and Considerations	21
4.4. Pull Web Service Filter Criteria	22
4.4.1. Filter Elements	22
4.5. Generic WSDL Example	25
5. BOD Specific Web Service Specifications	27
5.1. Overview	27
5.2. BOD Specific WSDLS	27
5.3. Benefits and Considerations	27
5.4. BOD Specific WSDL Example	28
6. Error Handling	31
6.1. HTTP Errors, SOAP Faults, and BOD Level Errors	31
6.1.1. General Principles	31
6.1.2. Spectrum of Error Types	31
6.1.3. HTTP Errors	32

6.2. SOAP Faults	33
6.2.1. Sample Error Cases	35
6.3. Application Level Errors	37
7. Security	39
7.1. Overview	39
7.2. WS-Security SOAP Header	39
7.3. Authentication	40
7.3.1. Username and Password	40
7.3.2. The Username element	41
7.3.3. Plain Text Password	41
7.3.4. Password Digest	41
7.4. Security Error Handling	42
STAR Interoperability Rules	45

List of Figures

2.1. Message Structure	6
2.2. Manifest	6
2.3. Process Message	10
2.4. ProcessMessage with Errors	10
2.5. ProcessMessage with Application System Errors	10
2.6. Successful PutMessage Sequence	12
2.7. PullMessage Structure	12
2.8. Successful PullMessage Operation	14
2.9. Payload Manifest	14
3.1. One-way Asynchronous Communication	18
3.2. Two-way Asynchronous Communication	19
4.1. PullMessage Filter Type	22
4.2. Generic Transport	25
4.3. Generic Payload Element Definition	25
4.4. Generic Element	26
5.1. WSDL Directory Structure	28
5.2. BOD Specific Service and Operations	29
5.3. BOD Specific Process Message Definition	29
5.4. BOD specific strongly typed payload	29
6.1. Spectrum of Error Types by Communication Mechanism	32

List of Tables

6.1. STAR Standard Soap Faults 33

List of Examples

2.1. Sample STAR Web Service Message	7
2.2. SOAP Body Message	9
2.3. SOAP Message	11
2.4. SOAP Message with Filter	13
4.1. Sample Generic Message	26
6.1. Sample SOAP Fault	34
7.1. Sample of WS-Security	40
7.2. WS-Security Username and Password	40
7.3. Username Element	41
7.4. Dealer Number	41
7.5. Unique ID that Identifies Dealer	41
7.6. Combination Dealer Number and ID	41
7.7. Plain Text Password	41
7.8. Password Digest	42

Preface

I.I. Purpose

The purpose of this document is to provide specifications and implementation guidelines for the STAR Web Service components.

This document is broken into the following sections for easier navigation:

- Preface - Overview of the specifications and background
 - Introduction - Background and general document overview
- Part I - STAR Level 1
 - Interface Specifications
 - Communication Patterns
 - Reliable Messaging
 - Error Handling
 - Security
- Part II - STAR Level 2 (Still in development)
 - WS-Addressing
 - WS-ReliableMessaging
 - Attachments - MTOM
 - Security
 - Applying Policy

I.II. SCOPE

This document covers the STAR Web Services interfaces specifications including the WSDL, message packaging, web methods, and different communication patterns. It also covers the STAR Web Services security specifications, based on OASIS WS-Security 1.0. This document does not address Identity, Authentication, Privacy, Content Integrity, Non-Repudiation and Trusted Timestamps. Versioning, Policy and Reliable messaging are also covered.

The following items have been defined as out of scope:

- Non-repudiation will be discussed under Auditing in a future release of these guidelines.

- Authorization, Trust Models and Attack Prevention are out of the scope for this release of the STAR Transport Guidelines and may be discussed in future releases of this guideline.
- Intermediaries, message routing, and other approaches to enhance or optimize the communication are also out of the scope of this document.

Note

This document is still under development. STAR Level 1 requirements have been added, but additional changes may be necessary. The namespace will not change as additional requirements are added. This document is expected to stabilize during the latter half of 2009.

I.III. AUDIENCE

This document is intended for application developers and application architects developing STAR Web Services interfaces.

I.IV. BACKGROUND

Web Services provide a standard means of interoperating between different software applications, running on a variety of platforms. Interoperability is achieved by using standard communication protocols that are platform neutral such as HTTP and XML to transport messages through the Internet. SOAP, Simple Object Access Protocol, is the main specification that describes how messages should be packaged in XML format. SOAP was submitted to the W3C in 2000 by IBM, Microsoft, UserLand, and Development. Other specifications work hand-in-hand with SOAP to provide complementary features such as WSDL (Web Service Description Language) to describe the interfaces and their bindings to communication protocols. And, UDDI (Universal Detection Discovery and Integration) to provide a registry service for service providers.

WS-I.org is the organization taking the responsibility of ensuring interoperability between the different Web Services implementations. In 2006, the organization published the WS-I Basic Profile 1.1, and this is the version that STAR is basing its web services guidelines on. The Basic Profile is based on the SOAP 1.1 specifications and describes SOAP bindings for HTTP only at this time. Bindings to other protocols such as TCP and SMTP are starting to emerge and might be included in a future version of the specifications.

I.V. SERVICE PROVIDER REQUIREMENTS

In order for a service provider to be able to receive and process requests and send responses back it must satisfy the following high level requirements detailed in other guidelines documents:

- Must have a fixed URL or IP address that is publicly accessible on the Internet.
- Must have the server software and infrastructure required to parse and process incoming messages.
- Security infrastructure to protect the publicly accessible servers as defined by Dealer Infrastructure guidelines or corporate security policy.

- Must have queuing facility to queue response messages if immediate delivery to the client is not possible (either disconnected client or a communication problem).

STAR defines eight security requirements:

- Business Authentication
- Party Authentication
- Privacy/Confidentiality
- Source and Target Authentication
- Source Only Authentication
- System Authentication
- Unique Party Identification

I.VI. COMMUNICATION PATTERNS OVERVIEW

This section provides an overview of the different communication patterns described in this document. For more detailed description, please refer to Communication Patterns chapter in the document.

Synchronous vs. Asynchronous

Synchronous communication refers to sending a message to a service provider and receiving a response within a short timeout period (recommended timeout is 100 seconds) on the same connection. A synchronous method invocation of a web service maps to one HTTP request/response cycle and it is similar to the way web pages are requested using a browser.

Synchronous method invocations are used when a response needs to be received immediately, say, to display it to a user in an interactive transaction.

Asynchronous communication, on the other hand, refers to sending a message without waiting for a response. A response is sent in a separate communication back to the originator. The response might be generated after a few seconds, a few hours, or even a few days depending on the business rules.

Synchronous communication, due to its nature, adds more requirements on both the server and the client than asynchronous communication. The server **MUST** process the received message and return a response within the preset time window, or return an error message.

One-Way vs. Two-Way

In compliance with the WS-I Basic Profile 1.1, STAR currently uses HTTP as the underlying transport protocol for Web Services. And, thus, follows the same request-based model. In a request-based model, the client always initiates the communication and the server always sends the responses on the same TCP connection to the IP address from which the request originated. This model works well with web services

and especially with low-end clients that do not have a fixed IP address or the infrastructure to support inbound requests.

In a one-way, request-based communication model, only the service provider is required to have a fixed IP address and the necessary hardware and software to listen to incoming messages. The client, on the other hand, can be very simple and can use any type of Internet connection, even those that do not provide a static IP address.

Due to its low requirements on the client end, the request-based model is suitable for dealer-to-OEM communication.

To achieve a two-way communication model, the original one-way model is duplicated in reverse: the client exposes the same set of web services and becomes a service provider too. This way, both sides are clients and service providers at the same time and they both can initiate requests to the other side. Based on business requirements and the agreement between the two parties, the client might choose not to implement the full set of functionality as the server to keep the implementation simple.

STAR Level One

This section describes the necessary components and pieces that all STAR Level 1 compliant implementations must implement.

Chapter 1, *STAR Web Services Overview*
Chapter 2, *Common Components*
Chapter 3, *Communication Patterns*
Chapter 4, *Generic Web Services Specifications*
Chapter 5, *BOD Specific Web Service Specifications*
Chapter 6, *Error Handling*
Chapter 7, *Security*

Chapter 1. STAR Web Services Overview

Table of Contents

1.1. Background	3
1.2. STAR Web Services Types	3
1.3. Web Service Interoperability Requirements	3

1.1. Background

The specifications define a set of methods and data types to facilitate exchanging synchronous and asynchronous messages using one-way or two-way communication models. This section describes these types and methods and explains how and where they apply.

This version of the specifications uses the following XML namespace to identify its types, methods and schemas:

<http://www.starstandards.org/webservices/2009/transport>

1.2. STAR Web Services Types

STAR supports two styles of WSDL.

- Generic Transport - This transport can handle any type of payload. It is up to the implementer to determine the type of payload being sent and received and act accordingly. One end point is used to process all transport requests.
- BOD Specific- This transport follows in line with the more traditional web service, as it expects a specific type of payload to be sent and a specific type to be returned. Multiple end points are needed to handle different types of BODS.

The type selected will depend on the requirements of the implementer. Some may choose to implement one or the other, and some may choose to implement both. A generic out-facing transport and possibly an internal BOD Specific transport to handle internal communications.

1.3. Web Service Interoperability Requirements

In order to ensure that the BOD Specific Web Service and the Generic Transport WSDL can exchange STAR BODs and interoperate, the SOAP Envelope and content must adhere to the same structure. The following items must much exactly:

- Element and attribute names in the Soap Envelope must much

- The structure of the SOAP Message being sent must match
- The STAR Manifest and STAR Payload must match
- Where the items appear within the Soap Envelope must match
- Occurrence constraints must match between the WSDLs. If something is required in one it must be required in the other
- If a field is optional in one it must be optional in the other



STAR Level 1 Requirement

STAR1015: STAR BOD Specific and Generic Transports must be message level interoperable.

As long as the message produced by the WSDL is the same between both services, the styles can communicate with each other. To help keep these aligned, STAR uses an XSLT Style Sheet to generate the sample STAR Transport 2005 and BOD Specific WSDL templates included with the STAR Schema Repository. If changes are made to the manifest or payload these will automatically appear in both the Generic and the BOD Specific WSDLs.

The WS-I profiles define standards for interoperability that make it easier to ensure that web services and clients can work together across varied platforms and implementations. STAR web services must conform to the WS-I Basic Profile 1.1 for interoperability and include conformance claims in the WSDL.



STAR Level 1 Requirement

STAR1001: All web services must be compliant to the rules and specifications outlined by the WS-I Basic Profile.

STAR1002: Appropriate compliance markers are required as specified by the WS-I Conformance Claim Attachment Mechanisms document.

Chapter 2. Common Components

Table of Contents

2.1. Overview	5
2.2. Message Packaging	5
2.2.1. Notes Regarding Payloads and Attachments	7
2.3. Namespaces	7
2.4. Web Methods	8
2.4.1. ProcessMessage	8
2.4.2. PutMessage	11
2.4.3. PullMessage	12
2.5. The payload Manifest SOAP Header	14

2.1. Overview

Regardless of whether a Generic Transport or a BOD Specific transport is being implemented, the overall message packaging will be the same. As was discussed earlier, the two transport mechanisms have to be inter operable at the messaging level. The following sections describe the message architecture that applies to both transport methods.

2.2. Message Packaging

The STAR Web Services transport was designed to provide a platform for secure and reliable delivery of any type of content in a standardized manner. The chosen architecture neither precludes nor requires attachments outside the body of the SOAP message for transportation of content. The chosen packaging methodology is well supported by all major Web Services toolkits and infrastructures and meets STAR's transport requirements.

The STAR Payload schema defines a package structure that provides additional features such as a standard way of packaging multiple contents (STAR BODs, XML documents, binary data, etc) in one payload and a message manifest that describes the contents of a message. The figure below shows the structure of a valid STAR Web Services message.

The first element under the SOAP:Body is the web method name. Three methods are defined: *ProcessMessage*, *PutMessage*, and *PullMessage*. Within the method element is the payload element, the primary element that encapsulates all transported payloads. The payload element contains one or more content elements, each of which encapsulates one and only one content element, such as a STAR BOD. The payload and content elements provide a standard format for transporting one or more XML documents inside the SOAP Body.

Figure 2.1. Message Structure

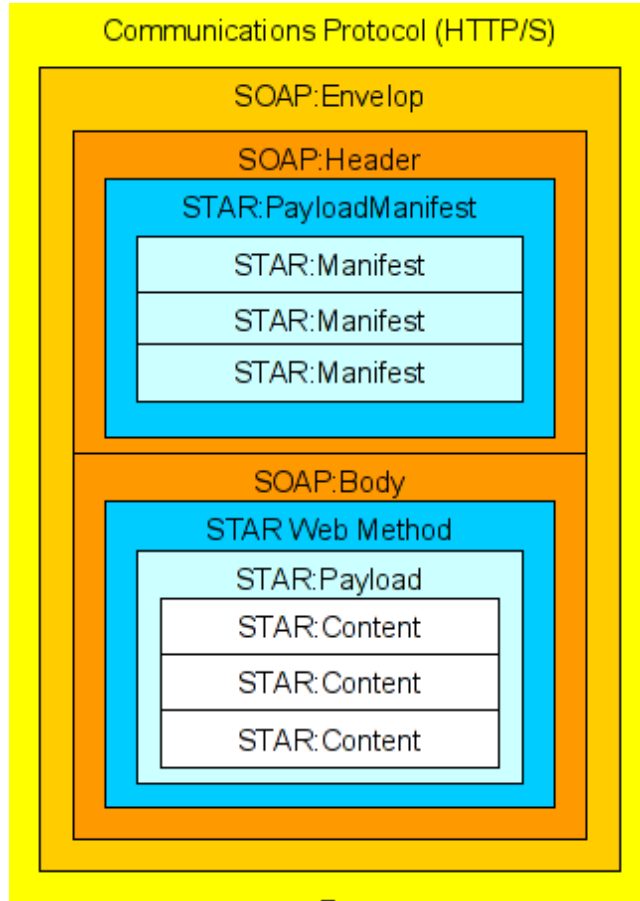
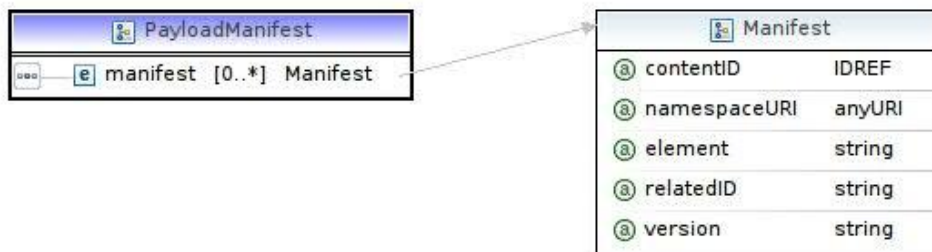


Figure 2.2. Manifest



In the SOAP:Header, STAR defines a payloadManifest element, which contains one or more manifest elements. Each manifest element corresponds to one content element in the SOAP Body and describes its contents. The payloadManifest and manifest elements provide a table of contents for the message.

The following sample shows the structure of the STAR Web Services message, including the location of the payloadManifest, and the star payload elements.

Example 2.1. Sample STAR Web Service Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:starws="http://www.starstandard.org/webservices/2009/transport">
  <soapenv:Header>
    <starws:payloadManifest>
      <!--Zero or more repetitions:-->
      <starws:manifest contentID="?" namespaceURI="?"
        element="?" relatedID="?" version="?" />
    </starws:payloadManifest>
  </soapenv:Header>
  <soapenv:Body>
    <starws:ProcessMessage>
      <!--Optional:-->
      <starws:payload>
        <!--Zero or more repetitions:-->
        <starws:content id="?">
          <!--You may enter ANY elements at this point-->
        </starws:content>
      </starws:payload>
    </starws:ProcessMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

2.2.1. Notes Regarding Payloads and Attachments

The decision was made to avoid dependency on Attachments. The currently defined interface specification neither requires nor prohibits attachments. While the overall message structure may not need to change, additional attributes or elements may need to be added to support the evolving web services attachments specifications in the future.

2.3. Namespaces

To avoid repetition and simplify the XML code snippets used in this document, the following namespace declarations will be used throughout this document:

uiPrefix	Description	Namespace
wsse	WS-Security	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	Utility Elements	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wsdl	WSDL 1.1	http://schemas.xmlsoap.org/wsdl/
soapbin	WSDL SOAP Binding	http://schemas.xmlsoap.org/wsdl/soap/
httpbin	WSDL HTTP Binding	http://schemas.xmlsoap.org/wsdl/http/

uiPrefix	Description	Namespace
mime	WSDL MIME Binding	http://schemas.xmlsoap.org/wsdl/mime/
soap	SOAP 1.1 Envelope	http://schemas.xmlsoap.org/soap/envelope/
xsi	Schema Instance	http://www.w3.org/2001/XMLSchema-instance
xs	XML Schema	http://www.w3.org/2001/XMLSchema
ds	XML Signature	http://www.w3.org/2000/09/xmldsig
xenc	XML Encryption	http://www.w3.org/2001/04/xm-lenc
starws	STAR Web Services	http://www.starstandard.org/web-services/2009/transport
oa	OAGIS	http://www.openapplications.org/oagis
oa9	OAGIS Version 9	http://www.openapplications.org/oagis/9
starbod	STAR BODs	http://www.starstandards.org/STAR
star5	STAR Version 5 BODs	http://www.starstandard.org/STAR/5
tns	This Name Space	Various
wsp	WS-Policy	http://www.w3.org/ns/ws-policy
wsa	WS-Addressing	http://www.w3.org/2005/08/addressing
wsrcm	WS-ReliableMessaging	http://docs.oasis-open.org/ws-rx/wsrcm/200608
wsam	WS-Addressing Metadata	http://www.w3.org/2007/05/addressing/metadata

2.4. Web Methods

Three methods are defined to cover the different types of communications supported by the guidelines. ProcessMessage, PutMessage, and PullMessage. The following sections describe these methods in more detail.

2.4.1. ProcessMessage

This is the method to use for synchronous communication. It takes a payload element as an input, processes it, and returns a result payload all within one HTTP cycle. After invoking this method, the client

keeps the connection open waiting for the response. If a response is not received within a predetermined timeout period, the method is considered to have failed.

In certain situations this method might return a SOAP fault element instead of a payload element. For example, if the sender could not be authenticated or the message is not well formed. Fault Codes are described in Chapter 6, *Error Handling* for more information. Errors related to business rules, on the other hand, **MUST NOT** be returned as SOAP faults, but returned using the suitable BOD.



STAR Level 1 Requirement

STAR1016: Application level error messages **MUST NOT** be returned with a SOAP Fault, and **MUST** be returned using the appropriate BOD.

Example 2.2. SOAP Body Message

Request:

```
<soapenv:Body>
  <starws:ProcessMessage>
    <!--Optional:-->
    <starws:payload>
      <!--Zero or more repetitions:-->
      <starws:content id=""?>
        <!--You may enter ANY elements at this point-->
      </starws:content>
    </starws:payload>
  </starws:ProcessMessage>
</soapenv:Body>
```

Response:

```
<soapenv:Body>
  <starws:ProcessMessageResponse>
    <!--Optional:-->
    <starws:payload>
      <!--Zero or more repetitions:-->
      <starws:content id=""?>
        <!--You may enter ANY elements at this point-->
      </starws:content>
    </starws:payload>
  </starws:ProcessMessageResponse>
</soapenv:Body>
```

The following sequence diagrams show the message exchange sequences for different scenarios.

Figure 2.3. Process Message

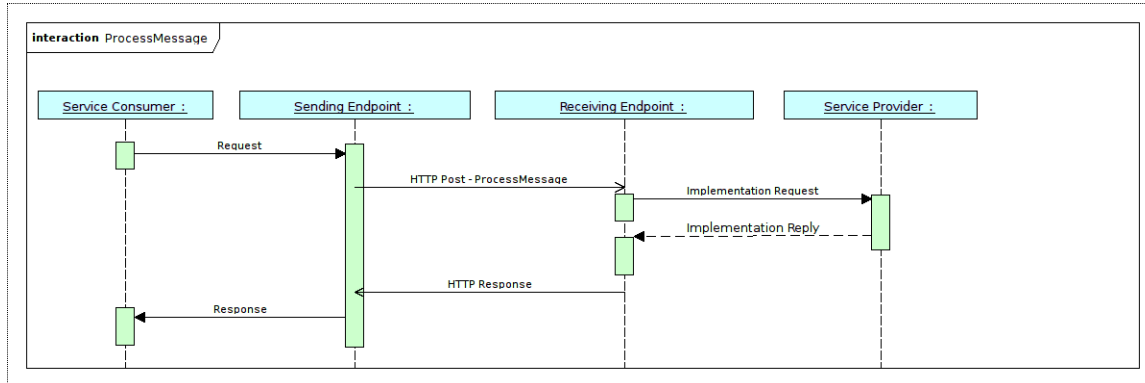


Figure 2.4. ProcessMessage with Errors

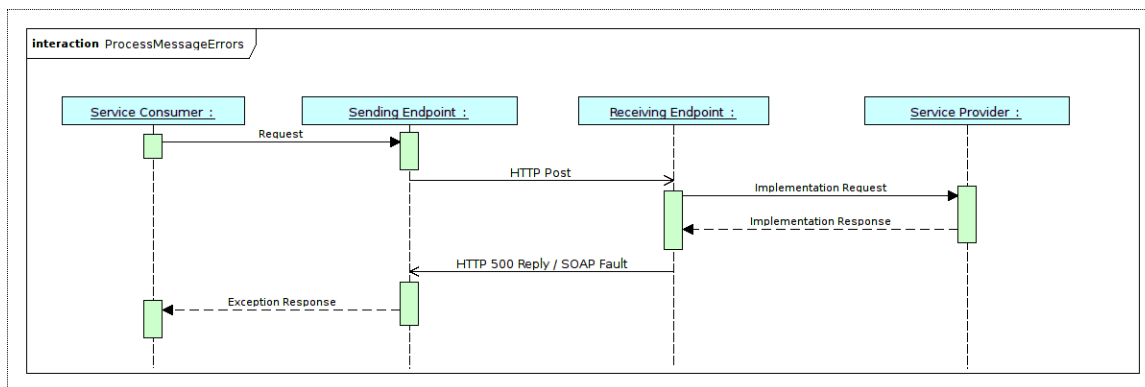
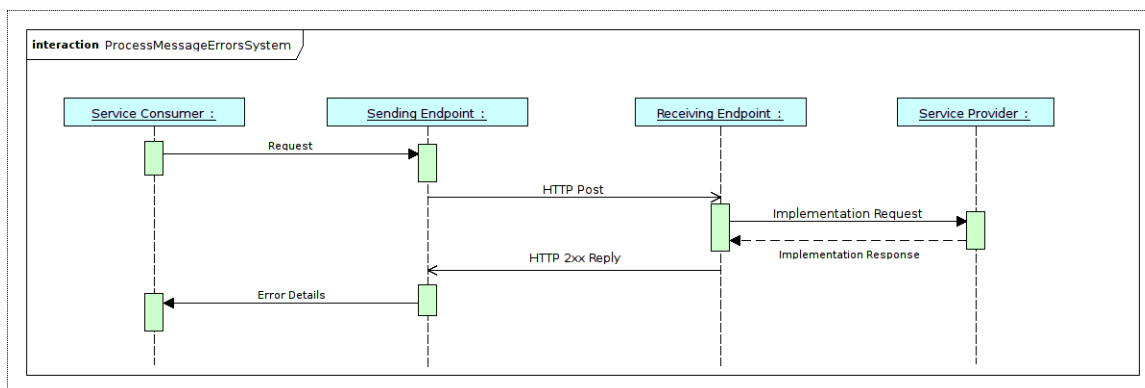


Figure 2.5. ProcessMessage with Application System Errors



The sequence diagrams that describe the error process are the same whether the PutMessage or PullMessage operation is being implemented instead of the ProcessMessage operation. The processes work the same whether a generic transport or a BOD specific transport is being implemented.

2.4.2. PutMessage

The PutMessage web method is used for asynchronous communication. It accepts a payload element as an input parameter, and returns nothing. Typically, PutMessage is used for messages that do not generate a response or for situations where a response is not returned immediately. The response can be retrieved later by calling PullMessage. The input payload for PutMessage must contain one or more elements.

Although PutMessage does not return any value to the caller, it can return SOAP faults to indicate that the 'put' process was not successful. This typically happens in situations where the message could not be parsed or persisted on the server side. For example, if the SOAP envelope is corrupted and the server can not extract the payload or the sender information then a SOAP fault must be returned on the same connection to inform the sender of the error. Note that business level errors such as invalid values in a BOD should not be returned as SOAP faults, but instead are returned asynchronously (not on the same HTTP connection) in a response BOD that describe the error details. SOAP faults are reserved for errors that prevent the correct parsing or persistence of the message on the server.

Example 2.3. SOAP Message

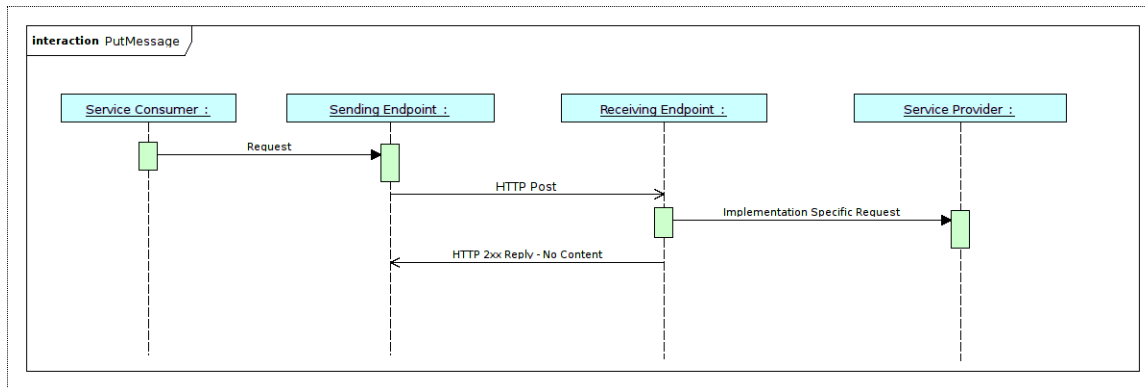
Request:

```
<soapenv:Body>
  <starws:PutMessage>
    <!--Optional-->
    <starws:payload>
      <!--Zero or more repetitions-->
      <starws:content id="?">
        <!-- You may enter ANY elements at this point-->
      </starws:content>
    </starws:payload>
  </starws:PutMessage>
</soapenv:Body>
```

Response:

```
<soapenv:Body>
  <starws:PutMessageResponse/>
</soapenv:Body>
```

In a one-way communication pattern, the client uses PutMessage to send a request to a service provider then sends another request using PullMessage (described in the next section) to pull the response. On the other hand, in a two-way communication pattern, the request is sent using PutMessage, and the response is returned using another PutMessage in the reverse direction.

Figure 2.6. Successful PutMessage Sequence

2.4.3. PullMessage

PullMessage is used to retrieve contents from the service provider. The contents can be:

Responses to previous contents (a BOD for example) submitted using PutMessage.

Responses to previous contents submitted using ProcessMessage but could not be delivered back to the requester due to communication or other errors.

Contents that originate from the service provider.

If the client is also a service provider, as in the two-way communication model, PullMessage is not required since both parties can communicate back and forth using PutMessage. However, the parties might choose to still use PullMessage in certain situations.

The service provider must keep track of contents that are deemed to have been received by the client to avoid resending. The client may receive duplicates during error recovery.



STAR Level 1 Requirement

STAR1017: The service provider must keep track of contents that are deemed to have been received by the client to avoid resending.

STAR1020: The client must be able to handle duplicate messages from a service provider.

Figure 2.7. PullMessage Structure

Filter Criteria:

Beginning in 2008 the PullMessage service was extended to support filtering with the addition of the filter complex type and the maxItems attribute. The filter type allows the requesting party to specify crite-

ria that the responding party must apply when selecting the BODs to be returned in the pull message response. The maxItems attribute can be used to limit the number of BODs that will be returned, regardless of whether or not they satisfy the filter criteria.

For example, the filter criteria and maxItems parameters could be used to satisfy the following request:

"Retrieve all AcknowledgePartsOrder BODs queued for sending in the last 24 hours. Send a maximum of 20 BODs"

A detailed description of the filter component can be found in section 4.5.

Returns:

This operation returns a payload object that carries one or more elements. Or, it returns an empty response with no payload element if there are no queued contents to return.

Example 2.4. SOAP Message with Filter

Request:

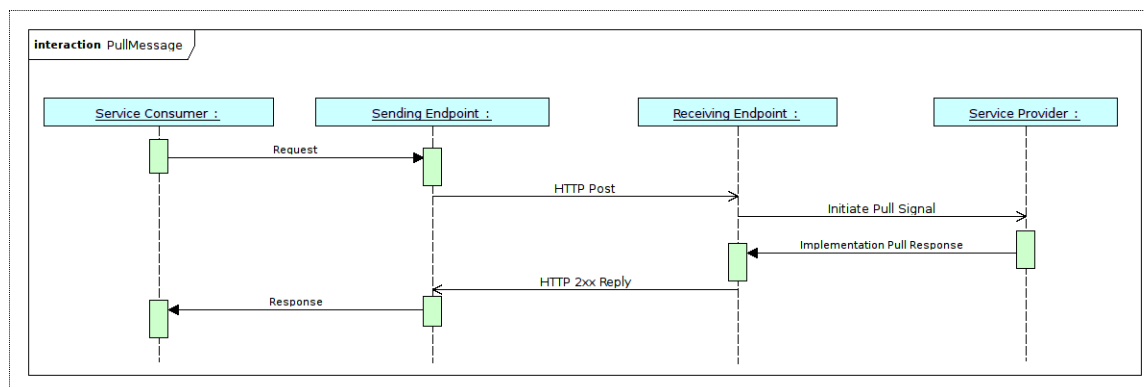
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tran="http://www.starstandard.org/webservices/2009/transport">
  <soapenv:Header/>
  <soapenv:Body>
    <tran:PullMessage maxItems="?">
      <!--Optional:-->
      <tran:filter>
        <!--Optional:-->
        <tran:filterConnection connectionID="?" destroy="?"/>
        <!--Optional:-->
        <tran:receiptIDs>
          <!--1 or more repetitions:-->
          <tran:receiptID?</tran:receiptID>
        </tran:receiptIDs>
        <!--Optional:-->
        <tran:filterCriteria>
          <!--1 or more repetitions:-->
          <tran:criterionList>
            <!--1 or more repetitions:-->
            <tran:criterion>
              <!--Optional:-->
              <tran:verb tran:operation="?"></tran:verb>
              <!--Optional:-->
              <tran:noun tran:operation="?"></tran:noun>
              <!--Optional:-->
              <tran:applicationID tran:operation="?"></tran:applicationID>
              <!--Optional:-->
              <tran:partyID tran:operation="?"></tran:partyID>
              <!--Optional:-->
              <tran:startDateTime tran:operation="?"></tran:startDateTime>
              <!--Optional:-->
              <tran:endDateTime tran:operation="?"></tran:endDateTime>
              <!--Optional:-->
              <tran:pullStatus tran:operation="?"></tran:pullStatus>
              <!--Optional:-->
              <tran:communicatorID tran:operation="?"></tran:communicatorID>
              <!--Zero or more repetitions:-->
              <tran:predefined tran:operation="?"></tran:predefined>
            </tran:criterion>
          </tran:criterionList>
        </tran:filterCriteria>
      </tran:filter>
    </tran:PullMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:

```

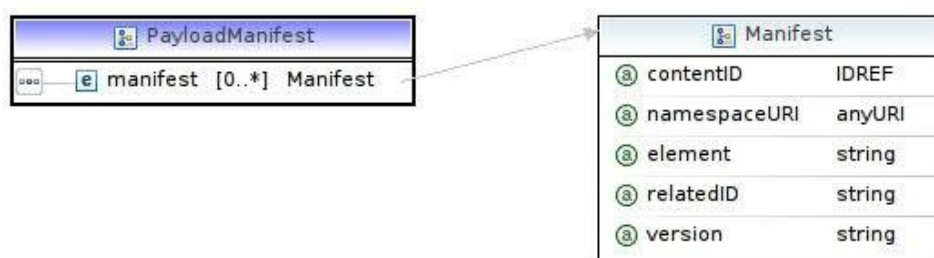
<soapenv:Body>
  <starws:PullMessageResponse>
    <!--Optional:-->
    <starws:payload>
      <!--Zero or more repetitions:-->
      <starws:content id="?">
        <!--You may enter ANY elements at this point-->
      </starws:content>
    </starws:payload>
  </starws:PullMessageResponse>
</soapenv:Body>

```

Figure 2.8. Successful PullMessage Operation

2.5. The payload Manifest SOAP Header

STAR defines a custom SOAP header to serve as a table of contents for the message. The payload manifest contains one manifest element for each content element in the SOAP body. The manifest provides an easy and fast way to identify the types of data in the message payload without parsing the whole message. This is useful for implementations that make routing decisions based on the contents of the message. And, it is especially useful if the body of the message is encrypted.

Figure 2.9. Payload Manifest

The manifest has the following attributes:

- *namespaceURI*: (Required) - This attribute contains the namespace URI of the XML element in the corresponding content in the SOAP body.
- *element*: (Required) - This attribute contains the local name of the XML element in the corresponding content in the SOAP body.
- *contentID*: (Required) This attribute should be populated with the ID of the corresponding *content* element. This attribute, along with the *id* attribute of the *content* element is used to match the *manifest* to its corresponding content element.
- *version* (Optional) - When the payload content is a BOD, this attribute contains the version number of the noun's schema used to validate the BOD, for example, 3.01. DTS files use the interfaceVersion of the file. For BOD content and DTS attachments this attribute is required.



STAR Level 1 Requirement

STAR1018: A SOAP Header **MUST** contain one **manifest element** for each **content element** in the SOAP body.

STAR1019: A **manifest** is **REQUIRED** to have **namespaceURI**, **element**, **contentID**, and **version** attributes. Even though version is listed as optional it is **REQUIRED** for STAR BOD and DTS transports.

Chapter 3. Communication Patterns

Table of Contents

3.1. One-Way Communication	17
3.1.1. One-Way Synchronous Communication	17
3.1.2. One-Way Asynchronous Communication	17
3.2. Two-Way Communication	18
3.2.1. Two-Way Synchronous Communication	18
3.2.2. Two-Way Asynchronous Communication	18

3.1. One-Way Communication

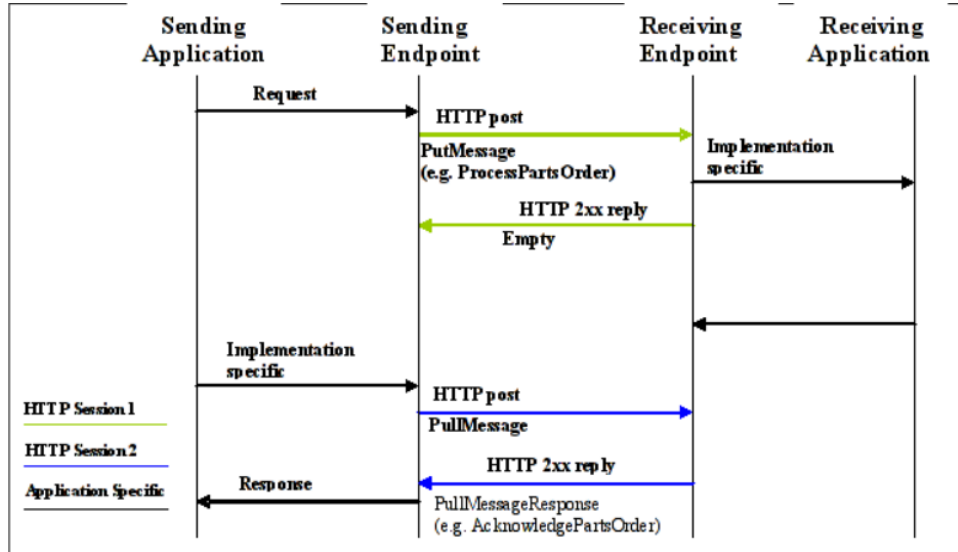
In the context of this document, one-way communication refers to the communication pattern in which only one of the communicating parties, the service provider (a.k.a. the server), has an addressable endpoint and the ability to receive and process incoming Web Services messages; and the other parties, the service consumer (a.k.a. the client), can send Web Services requests and receive their response in one HTTP cycle, but does not have an addressable endpoint to receive incoming messages initiated by another party. In this communication pattern, messages always originate from the client to the server.

3.1.1. One-Way Synchronous Communication

The client uses `ProcessMessage` to achieve synchronous communication and receive a response immediately as shown in `ProcessMessage` sequence diagram earlier. Upon receiving the request, the server starts processing it while holding the connection with the client open until a response (or an error) is ready to be returned to the client on the open connection.

3.1.2. One-Way Asynchronous Communication

The client can also use `PutMessage` and `PullMessage` together to achieve asynchronous communication as shown in the figure below. In this pattern, the client must send a `PullMessage` request to receive contents queued at the server side. The client can either implement a polling service to periodically request contents from the server or send the requests only when contents are expected to be available for download, for example, through an event notification model, the details of which is out of the scope of this document.

Figure 3.1. One-way Asynchronous Communication

3.2. Two-Way Communication

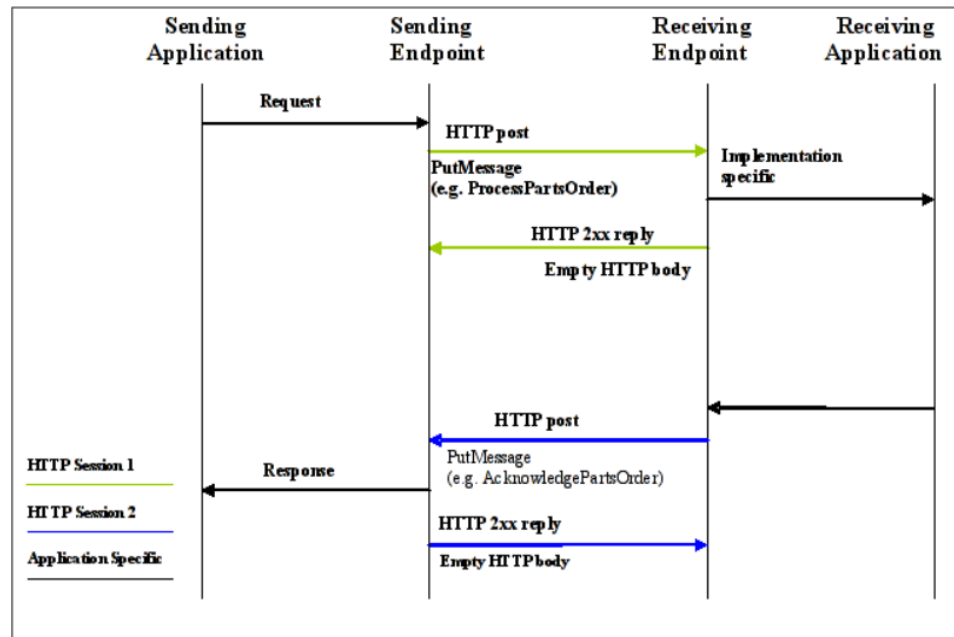
Two-way communication in the context of this document refers to the pattern in which both communicating partners have the ability to initiate and receive messages at the same time. This type of communication is possible if both parties satisfy the service provider requirements described in Service Provider Requirements section.

3.2.1. Two-Way Synchronous Communication

Synchronous communication is done the same way using ProcessMessage as it is done in the one-way pattern (see Figure 3.1, “One-way Asynchronous Communication”). The difference here is that both parties can initiate the requests and hold for a response. Business requirements and an agreement between the two communicating parties determine whether and when synchronous communication is appropriate versus asynchronous communication.

3.2.2. Two-Way Asynchronous Communication

Asynchronous communication changes a little bit from the way it is done in the one-way communication pattern. In the one-way approach, the client sends a request using PutMessage, and then sends another request using PullMessage to download the response from the server. In the two-way approach, the need for PullMessage diminishes and is replaced instead by PutMessage initiated by the server to the client as shown in the figure below.

Figure 3.2. Two-way Asynchronous Communication

Chapter 4. Generic Web Services Specifications

Table of Contents

4.1. Overview	21
4.2. Generic WSDL	21
4.3. Benefits and Considerations	21
4.4. Pull Web Service Filter Criteria	22
4.4.1. Filter Elements	22
4.5. Generic WSDL Example	25

4.1. Overview

The specifications define a set of methods and data types to facilitate exchanging synchronous and asynchronous messages using one-way or two-way communication models. This section describes these types and methods and explains how and where they apply.

4.2. Generic WSDL

The generic transport is considered to be a loosely typed WSDL, meaning that it does not fully describe all types of payloads that can occur. It provides meta data about the payload that shows up in the SOAP BODY based on information contained in the SOAP Header manifest, and the content element in the SOAP BODY. The generic transport does exactly what its name implies, allowing the sending and receiving of any type of payload in the soap body. You could technically send a BOD, UBL Message, DTS Transaction, Text, Binary Encoded, etc.

4.3. Benefits and Considerations

Benefits:

- Allows for loosely typed, and loosely coupled systems. Transport and Application are separate
- Changes to the data sent in the payload do not necessarily change the WSDL. Meaning the web service does not change because the schema changed.

Considerations:

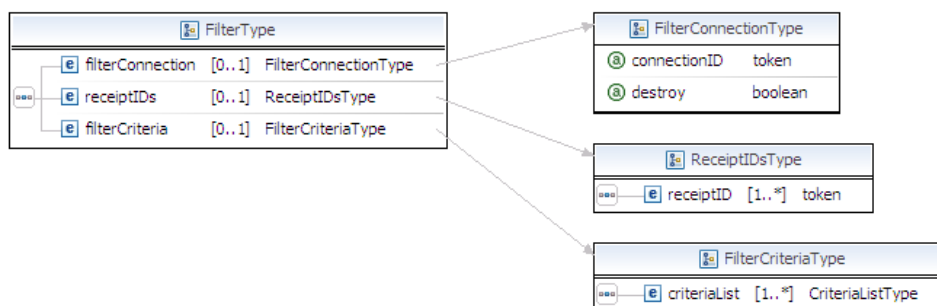
- All transport traffic is going through one end point. This could potentially have scalability issues depending on the amount of data that is coming into the system.
- Implementers will need to implement routing and extraction code in order to determine what to do with the payload received.
- Need to implement logic in order to handle contents that are not understood.

- Need to negotiate out of band using other services to describe what payloads are understood and handled by a particular trading partner.

4.4. Pull Web Service Filter Criteria

As discussed in Chapter 2, *Common Components*, the Pull web service was enhanced for 2008 with a Filter component. This component allows the service requestor to provide optional criteria that the service provider will use to restrict the number and types of BODs that will be returned in the response message. Additionally, filters can be defined as persistent, allowing them to be re-used across multiple Pull requests.

Figure 4.1. PullMessage Filter Type



Each service requestor and service provider must implement the code that provides support for the filter component. STAR will provide the specifications for the filter component, however each implementor will be free to choose the method in which they implement the functionality.

4.4.1. Filter Elements

The Filter component consists of the following three elements:

Element	Occurrence	Description
filterConnection	Complex Type	Used to define persistence for the filter
receiptIDs	Complex Type	Used by service requestor to confirm the receipt of each message requested
filterCriteria	Complex Type	List of filter criteria to apply to the pull request

The complete list of elements within the filter criteria component are shown below.

Item	Type	Description
PullMessage	Complex Type	
maxItems	Attribute	The maximum number of items to be sent. The service may send less than the number requested but should never send more than

Filter Elements

		the number requested in any one pulling session.
filter	Complex Type	
filterConnection	Element	
connectionID	Attribute	A unique connection id for the filter. Used during persistence of a filter.
destroy	Attribute	The destroy attribute of FilterConnection will be set to true when the client decides to destroy a persisted filter before all of its applicable messages have been pulled. If and when the client does pull all of the persisted filter's applicable messages, then the web service will automatically destroy the connection and return an empty pull response. If the client does not pull all of a persisted filter's applicable messages and does not explicitly destroy the persisted filter by setting the destroy attribute to true, then based on an agreed upon out-of-band policy, the web service will expire the persisted filter after X number of days.
recieptIDs	Complex Type	
receiptID	Element	An unbounded list of content ids that have been previously received since the last pull request.
filterCriteria	Complex Type	A list of filter criterias to be applied to pulling.
criteriaList	Complex Type	Criteria contains a unbounded list of filter criteria that can be applied to a queue. If included it is used to specify what should be retrieved. More than one criteria can be specified. Each criteria is it's own filter.
verb	Element	The OAGIS or STAR Verb. i.e. Process, Acknowledge, Notify, etc
operation	Attribute	Enumerated List: "and", "or", "not"
noun	Element	The OAGIS or STAR Noun for a particular BOD. i.e. PartsOrder,

Filter Elements

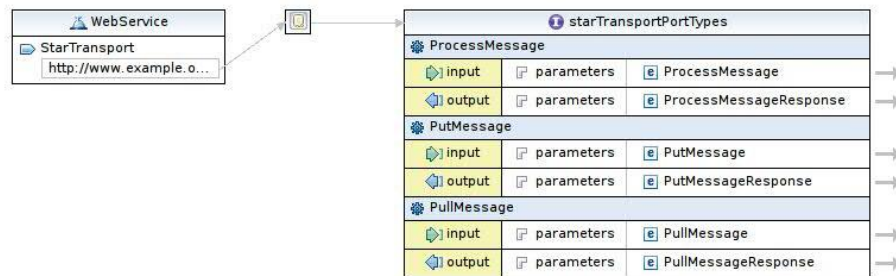
		CreditApplication, FinancialStatement, etc.
operation	Attribute	Enumerated List: "and", "or", "not"
serviceID	Element	identifies the particular service to or from which a message is being sent (e.g. Parts:Orders)
operation	Attribute	Enumerated List: "and", "or", "not"
partyID	Element	Assigning Organization Party Id
operation	Attribute	Enumerated List: "and", "or", "not"
startDateTime	Element	Indicates the beginning time/date range of messages to be retrieved during this pull session. Based on the time/date at which each message was originally queued for delivery.
operation	Attribute	Enumerated List: "and", "or", "not"
endDateTime	Element	Indicates the ending time/date range of messages to be retrieved during this pull session. Based on the time/date at which each message was originally queued for delivery.
operation	Attribute	Enumerated List: "and", "or", "not"
pullStatus	Element	The status of an item to be pulled. (i.e. Pulled, Ready, etc.)
operation	Attribute	Enumerated List: "and", "or", "not"
communicatorID	Element	Identifier of the party on behalf of which the pull call was submitted. This could be the ID of the calling party or it may be an alternate party if the pull request is being proxied by another service.
operation	Attribute	Enumerated List: "and", "or", "not"
predefined	Element	These are complex queries or queries that can't be represented using the current filter criteria.

		They may contain if then else logic, and are identified by a name. (i.e. GetWidgetsGreaterThan10)
operation	Attribute	Enumerated List: "and", "or", "not"

4.5. Generic WSDL Example

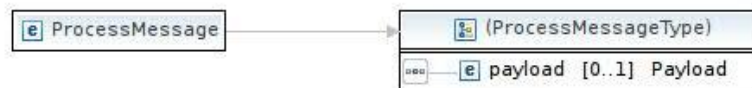
How is the generic transport implemented by STAR? As has been outlined in Chapter 2, *Common Components*, the generic transport will implement the Manifest, and Content elements in the Soap Header and Body respectively.

Figure 4.2. Generic Transport

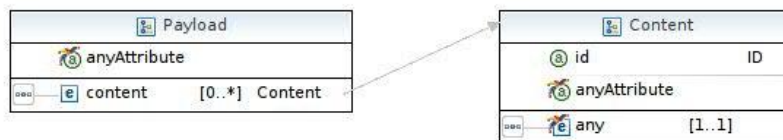


As is depicted in the figure below, the WSDL implements the ProcessMessage, PutMessage, and PullMessage methods and operations. The following examples will use the ProcessMessage method to indicate the structure of the SOAP Body. A sample Generic Transport WSDL can be found with the STAR Schema Repository.

Figure 4.3. Generic Payload Element Definition



The generic transport will use one common payload element definition. This is the Payload type. The payload type contains the definition for the content elements.

Figure 4.4. Generic Element

The Generic content element refers to a complex Type definition that defines an `xsd:any` as the content for the element. What this says, is that any type of XML or Text can be put here. It is not locked to a particular type of data to be sent or received. Information about the content is located in the Manifest elements and linked by an id. There may be unlimited number of content elements sent in the payload, and each links back to a particular manifest element.

The receiving web service would process the Manifest to determine what it actually received, and do any appropriate routing or processing of the payload contained within it.

Example 4.1. Sample Generic Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:starws="http://www.starstandard.org/webservices/2009/transport">
  <soapenv:Header>
    <starws:payloadManifest>
      <!--Zero or more repetitions:-->
      <starws:manifest contentID="?" namespaceURI="?" element="?" relatedID="?"
        version="?" />
    </starws:payloadManifest>
  </soapenv:Header>
  <soapenv:Body>
    <starws:ProcessMessage>
      <starws:payload>
        <!--Zero or more repetitions:-->
        <starws:content id="?">
          <!-- You may enter ANY elements at this point-->
        </starws:content>
      </starws:payload>
    </starws:ProcessMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Chapter 5. BOD Specific Web Service Specifications

Table of Contents

5.1. Overview	27
5.2. BOD Specific WSDLs	27
5.3. Benefits and Considerations	27
5.4. BOD Specific WSDL Example	28

5.1. Overview

Choosing either a BOD Specific transport or a Generic transport does have architectural implications. However, the choice does not have to be one or the other, both can be used together. The choice should be based on what best fits the overall architectural and system needs for the Web Services to be implemented.

5.2. BOD Specific WSDLs

What is a BOD Specific WSDL? A BOD Specific WSDL is a version of the STAR Web Services Transport specification that expects to send and receive only a specific type of BOD in its transaction life cycle. If it receives anything other than what it expects, it will send back a SOAP Fault to indicate that the wrong payload was sent. According to Russel Butek from IBM, "WSDL is the Web Services Description Language. Its charter is to describe an interface to a service as completely as possible. When you use `xsd:any`, you deviate from this intent of the WSDL".[Bukett 4] In other words, a Generic transport does not fully describe a web service; it leaves key information about the payload out. This has the advantage of allowing the transport to remain generic but shifts the determination of the content and what to do with the content received to another portion of the system.

A BOD Specific WSDL will fully describe all aspects of the Web Service's capabilities, including the type of payload that can be received and the expected responses. BOD specific WSDL is considered to be a strongly typed WSDL. More information in regards to strongly typed and loosely typed WSDL definitions can be found in the IBM Developerworks article, "Loosely typed versus strongly typed Web Services" by Andre Tost.

5.3. Benefits and Considerations

Benefits:

- BOD specific WSDL fully describes the Web Services Interface and the type of services it offers. The WSDL is considered to be strongly typed.
- BOD specific WSDL specifies clearly what is to be sent to the service and what is to be returned.

- BOD specific WSDL's are more compatible with existing development tools that generate code from the WSDL. These tools work best when they can describe the full capabilities, and not have to leave pieces to be filled in outside of their framework.
- Services using BOD Specific WSDL's do not require additional processing by the SOAP engine to figure out the type of payload being received.
- Data Validation of the payload can happen before it reaches the application, as it is validated by the type of content the Web Service expects to receive.

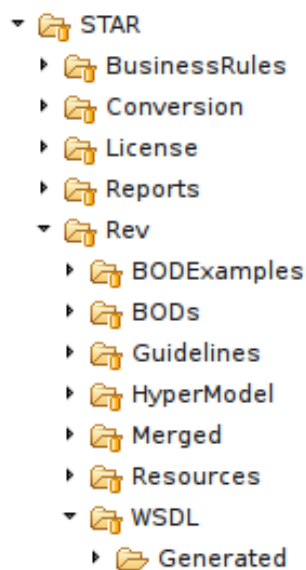
Considerations:

- Changes can create backward compatibility issues. If the strongly typed data in the WSDL changes or breaks compatibility, code that depends on the WSDL may need to be regenerated.
- Strongly typed interfaces require more logic upfront in the Transport dealing with the payload and parsing of the information. The amount depends on the size and complexity of the payload.
- There is a closer tie between the transport and application, potentially requiring closer testing between the two.

5.4. BOD Specific WSDL Example

So how does a BOD Specific WSDL look when implementing the STAR Web Services Transport? STAR includes with the XML Schemas for STAR 5 the BOD Specific WSDL for all the BODs. These are grouped by the recommended Verb and Noun pairing outlined in the Verb Usage Guidelines available on the STAR Website. The WSDL files may be found in the following directory:

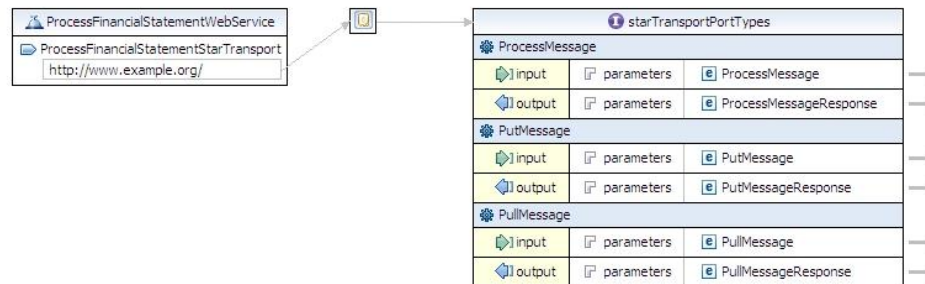
Figure 5.1. WSDL Directory Structure



There are roughly about 60 WSDL definition files, and these are automatically generated with the base information necessary for minimum compliance with the STAR Web Services transport. These WSDL files

do not implement any of the Security, or Reliable Messaging that may be needed by a particular implementation. They are provided as templates for users to update for their particular requirements.

Figure 5.2. BOD Specific Service and Operations



On the surface, there is little difference between the structure of the BOD Specific WSDL and a generic transport. You still have operations, transport types, bindings, messages, and services. The root soap body elements that carry the payload are still the same, and the manifest information that is transmitted is the same as well. These all have to be the same for both a generic and BOD specific WSDL to be able to interoperate with each other.

Differences start to show once you reach the definition of the elements that make up the operation as shown in the figure below Figure 19.

Figure 5.3. BOD Specific Process Message Definition



A BOD specific WSDL will refer to a very strongly typed definition for the payload element. This allows the WSDL to fully describe the type of content expected to be sent with the type of operation that is being invoked. Further, the payload element itself describes the type of BOD to be carried and the multiplicity of the content. It also describes where attachment data should occur and in what order the payload information should be sent.

Figure 5.4. BOD specific strongly typed payload



If the information that is sent does not appear in the order that is specified, a SOAP Fault is returned before the information ever reaches the receiving application. See the sample ProcessFinancialStatement BOD-Based output that could be generated by the CancelFinancialStatement WSDL file included with the STAR Schema Repository.

Resources for Works Cited:

Tost, Andre, "Loosely typed versus strongly typed Web Services", 02 Sep 2005, IBM Developer Works, 19 Dec 2007, <http://www.ibm.com/developerworks/webservices/library/ws-loosevstrong.html> [<http://www.ibm.com/developerworks/webservices/library/ws-loosevstrong.html>]

Butek, Russell, "Tip: xsd:any: A cautionary tale", 13 Dec 2005, IBM Developer Works, 19 Dec 2007, <http://www-128.ibm.com/developerworks/xml/library/ws-tip-xsdcaution.html> [<http://www-128.ibm.com/developerworks/xml/library/ws-tip-xsdcaution.html>]

Chapter 6. Error Handling

Table of Contents

6.1. HTTP Errors, SOAP Faults, and BOD Level Errors	31
6.1.1. General Principles	31
6.1.2. Spectrum of Error Types	31
6.1.3. HTTP Errors	32
6.2. SOAP Faults	33
6.2.1. Sample Error Cases	35
6.3. Application Level Errors	37

6.1. HTTP Errors, SOAP Faults, and BOD Level Errors

It is important to use HTTP errors, SOAP Faults, or BOD level errors consistently. However, this section acknowledges implementations may use different error mechanisms for the same types of errors.

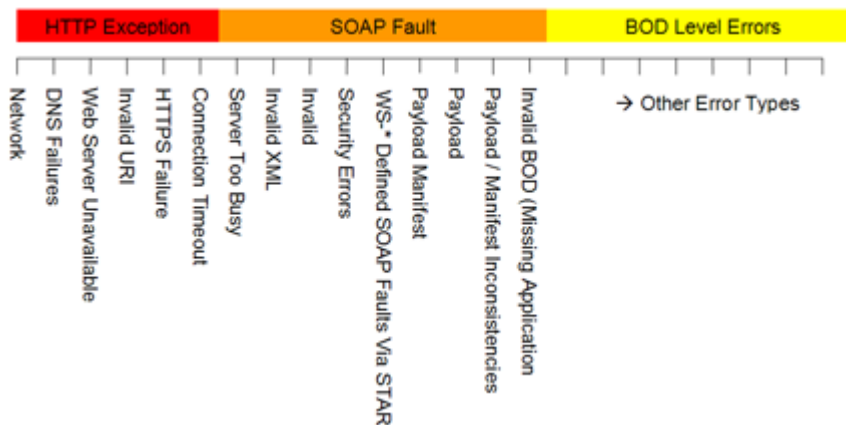
This section defines general guidelines between these error mechanisms and also refers to the appropriate STAR documentation with regard to BOD level error handling.

6.1.1. General Principles

- Across STAR transports, the mechanism of error reporting should be as consistent as possible.
- To align with the above principle, communicate as many errors as possible within BODs and minimize the amount of errors communicated in a transport specific way (such as SOAP Faults).
- Other principles specific to BOD level errors are detailed in the STAR Confirm BOD Implementation Guidelines.
- Implementations **MUST NOT** communicate errors using mechanisms other than those defined by STAR documents, or use error codes not defined or approved by STAR.

6.1.2. Spectrum of Error Types

The types of errors which can occur when a Web service client attempts to communicate with a Web service can be thought of as a spectrum. At one end of the spectrum are the pure transport related errors and at the other end are the errors resulting from the business processing of the BOD. The mechanisms used to communicate these errors differ. 4.2.2-1 depicts an example specific to the use of Web services over HTTP. Other transport level protocols may have other exceptions. Although many types of errors are shown, this list is not intended to be all-inclusive.

Figure 6.1. Spectrum of Error Types by Communication Mechanism

The general guidelines with the approach shown here are the following:

HTTP exceptions are truly transport specific. SOAP Faults include the following types of errors:

- True Web service transport specific errors (e.g., Invalid operation and server too busy)
- SOAP Faults defined as part of a Web service standard like WS-Security implemented by this document (e.g., FailedAuthentication)
- SOAP Faults due to manifest and payload inconsistencies
- SOAP Faults due to business level errors that could not be returned in a BOD.

BOD level errors are preferred over the previous transport specific errors; therefore, if the application area of the BOD is obtainable and the error is not a transport error, BOD level error handling **SHOULD** be used. Refer to the STAR Confirm BOD Implementation Guidelines for details on BOD level errors.

6.1.3. HTTP Errors

HTTP Errors are those that occur at the transport layer when trying to call the web service and the web service client should handle them according to WSI basic profile 1.1.

Web service client should check for the HTTP return code 200 to ensure the transaction that was attempted went through ok. If the client receives any HTTP return code other than 200, it should be handled accordingly. Typical situations an HTTP error could occur are as follows:

Common STAR HTTP Error Codes:

Description	Error Code
When the web server where the web service is being hosted is down or not available.	404
When the DNS server is unable to resolve the domain name in the Endpoint.	400 or 500
When the Endpoint of the web service is not available.	503

When the SSL Handshake Failure occurs between the Client and the Web Server.	500
When the Web Service does not recognize the request sent by the client.	502
When the HTTP Request time out occurs. (NOTE: This is not the same as the timeout error we see in the SoapFault.)	408

For a more details on all type HTTP error codes and details of each error code and description please refer to RFC2616 Section 10. [<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>]

6.2. SOAP Faults

SOAP faults are used to indicate error situations that prevent the successful delivery of STAR contents for subsequent parsing and processing. While this specification does not prevent using SOAP faults to communicate business level errors generated by business rules, this specification **RECOMMENDS** that these types of errors be returned using the proper BOD type such as a ConfirmBOD or the corresponding Acknowledge BOD.

If you send a message that was not successful you may get back a response containing a SOAP fault element which gives you status information, error information, or both. A SOAP Fault is similar to an Exception object in common development platforms in that it conveys information about a problem that prevents further processing. Some STAR-specific SOAP fault codes have been defined for common faults. All STAR implementations must understand these faults and handle them accordingly. Standard SOAP faults should be preferred over custom fault codes, such as when WS-Security or the WS-I Basic Profile define specific faults to be used. Below is a list of common faults that have been defined by STAR. When used, the fault code must be prefixed by “STAR:” and appear as in STAR:Invalid Structure.

Note: When sending a SOAP Fault the HTTP Status code needs to be set to 500, according to the WS-I Basic Profile.

Table 6.1. STAR Standard Soap Faults

Fault Code	Description
Duplicate Document	This code refers to a document that already exists. This may happen for a BOD such as ProcessPartOrder where the document identifiers to another existing parts order from the same dealer.
Not Authorized	This code occurs when the client attempts to perform an operation that is not authorized for the given action. This is not to be used for Authentication errors. Those should use the appropriate WS-Security SOAP Fault.
Server Error	An error (e.g. database server is down) on the server prevented the execution of the BOD. The client will have to resend the BOD at a later time.

Fault Code	Description
BOD Not Supported	The received BOD or BOD version is not supported by the receiver.
Invalid Structure	The structure of the BOD is not valid. For example, the BOD failed schema validation.
Invalid BODID	A BODID was missing or is Invalid.
Time Exceeded	The processing time will exceed the real time transaction allowed time. Must resend with a Put for batch processing, and pull to receive the message.

A SOAP Fault object contains the following elements:

- Fault code: Always required. The fault code must be a fully qualified name: it must contain a prefix followed by a local name. The SOAP specifications define a set of fault code local name values, which a developer can extend to cover other problems.
- Fault string: Always required. A human-readable explanation of the fault.
- Fault actor: Required if the SOAP Header object contains one or more actor attributes; optional if no actors are specified, meaning that the only actor is the ultimate destination. The fault actor, which is specified as a URI, identifies who caused the fault. For an explanation of what an actor is, see the actor attribute.
- Detail object: Optional element. The SOAP Fault object may contain a Detail object that gives details about the problem.

Below is an example of a SOAP message carrying a valid Fault element:

Example 6.1. Sample SOAP Fault

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Database server not available.</faultstring>
      <faultactor>http://localhost/Webservices/STAR/STARTransport.asmx</faultactor>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

SOAP 1.1 defines the following standard fault codes under the SOAP namespace ("http://schemas.xmlsoap.org/soap/envelope/"):

- Client: This code should be used when an error is found in the received message. The error could be anything from a corrupted message to a missing required element. This fault code indicates that the received message is the cause of the error and that the client is to blame.
- Server: This fault code indicates that a problem at the server prevented the processing of the message. The error could be anything from an overloaded server to a failing database.

These fault codes represent classes of errors rather than specific errors. SOAP 1.1 allows extending the fault codes using the period notation, however this practice is discouraged by the WS-I Basic Profile 1.0 to avoid the risk of potential name conflicts.



STAR Level 1 Requirement

STAR1009: All STAR Web Services are **REQUIRED** to understand and handle the STAR Specific SOAP Faults.

STAR1010: All STAR soap fault error codes are **REQUIRED** to be prefixed with STAR: and the appropriate STAR error code. i.e. **STAR:Invalid Structure**.

STAR1011: All STAR soap fault error codes are **REQUIRED** to appear in the standard SOAP:Fault block.

STAR1012: SOAP Faults are for Critical Processing errors only. Informational or warning errors **SHOULD NOT** be sent as a SOAP Fault.

Note that some specifications mentioned in this document define their own SOAP faults. For example, WS-Security defines a set of fault codes to address security related errors. These fault codes are described in the WS-Security section and should be used when appropriate.



STAR Level 1 Requirement

STAR1014: WS-Security errors must send the appropriate WS-Security SOAP Fault for the authorization being used.

6.2.1. Sample Error Cases

Below are examples of different error situations and valid responses that a service provider can reply with.

Error Case	Valid Response (ConfirmBOD or SOAP Fault)
Wrong ProcessMessage namespace	Fault: soap:Client
Wrong BOD namespace	Fault: soap:Client
Misspelled BOD root element	ConfirmBOD: BodNotSupported Fault: soap:Client
Invalid or missing <Task> element	Fault: soap:Client Fault: wsse:FailedAuthentication ConfirmBOD: BodNotSupported ConfirmBOD: FieldMissing

Sample Error Cases

	ConfirmBOD: InvalidBod
Missing <ReferenceId>	Fault: soap:Client ConfirmBOD: FieldMissing
Missing <DealerNumber>	Fault: soap:Client Fault: wsse:FailedAuthentication ConfirmBOD: FieldMissing
Invalid dealer number	Fault: wsse:FailedAuthentication ConfirmBOD: InvalidValue
Missing <BODId>	Fault: soap:Client ConfirmBOD: FieldMissing
Message too old (wsse:Security\wsu:Timestamp\wsu:Expires has expired)	wsu:MessageExpired
Missing Application Area	Fault: soap:Client Fault: wsse:FailedAuthentication ConfirmBOD: InvalidBod
Missing Data Area	Fault: soap:Client ConfirmBOD: InvalidBod
Invalid User ID in wsse:Security header	wsse:FailedAuthentication
Wrong password in wsse:Security header	wsse:FailedAuthentication
Corrupted XML	HTTP/1.1 400 Bad Request (specified by WS-I Basic Profile)
Wrong SOAP Action HTTP header	soap:Client
Wrong SOAP Action in HTTP header and wsa:Action	soap:Client

Misspelled wsse:Security namespace

wsse:SecurityTokenUnavailable

6.3. Application Level Errors

Application level errors are those that occur once the payload has made it into the application for processing. The BOD that returns these errors could either be a Acknowledgement or a Confirm depending on the verb that was used to send the BOD. If a Process verb is used, then an Acknowledgement Verb is the appropriate response. Confirm BOD could technically be used in almost any situation, but it is an OAGi recommendation that application level errors be handled where possible by the corresponding response Verb.

The following messages may occur at a ConfirmBOD or SOAP Fault level. This depends on how the implementation is architect-ed on the back end system. While the ConfirmBOD can send back Warnings, SOAP Faults are restricted to errors that stop processing of the message. Warnings are not included in the list for this reason. There are also some concerns about warnings on how these should be handled from an interoperability standpoint.

All of the codes listed in the following table are to be treated as ERRORS.



STAR Level 1 Requirement

STAR1013: ConfirmBOD reason codes that are sent at the Warning or Informational status, should not trigger a resending of the BOD.

Description	Code
A document already exists. This may happen for a BOD such as ProcessPartsOrder where the document identifiers to another existing parts order from the same dealer.	Duplicate Document
One or more required data elements have invalid values.	Invalid Required Value
The operation that cannot be performed, such as Change or Cancel based on the receiver's business rules and the condition of the document. For example, the part order has already been shipped therefore the order cannot be cancelled.	Cannot Perform
One or more required fields are missing.	Required Field Missing
An error (e.g. database server is down) on the server prevented the execution of the BOD. The client will have to resend the BOD at a later time.	Server Error
The client attempts to perform an operation that is not permitted. An example of when this may occur is if the dealer attempts to order a part when their account is placed on hold. This is to be used for authorization errors	Not Permitted
	BOD Not Supported

Application Level Errors

The received BOD or BOD version is not supported by the receiver.	
The structure of the BOD is not valid. For example, the BOD failed schema validation.	Invalid Structure

Chapter 7. Security

Table of Contents

7.1. Overview	39
7.2. WS-Security SOAP Header	39
7.3. Authentication	40
7.3.1. Username and Password	40
7.3.2. The Username element	41
7.3.3. Plain Text Password	41
7.3.4. Password Digest	41
7.4. Security Error Handling	42

7.1. Overview

The following sections define the implementation details to meet the Star Transport Guidelines security requirements when using Web Services.

The following specifications are used to accomplish secure web services communication until further clarifications and standards emerge from the Web Services Security technical committee in Oasis:

1. HTTPS: Provides a secure transport channel
2. Web Services Security: SOAP Messaging Security V1.0: Provides the framework for SOAP messaging security.
3. Web Services Security: Username Token Profile V1.0: Describes user authentication tokens.

The security methods described in this section can be applied to all the web services methods mentioned earlier on both requests and responses. Communication partners will need to agree on which security methods to use and on which types of communication. The choice will also be affected by business rules, performance and information sensitivity. As a base standard all STAR endpoints and clients **MUST** send information encrypted using HTTPS and comply with the security requirements outlined by the WS-I Basic Security Profile 1.0.



STAR Level 1 Requirement

STAR1004 : All implementations are **REQUIRED** to send information over HTTPS.

STAR1008 : All services and clients must be compliant to the general Security requirements outlined by the WS-I Basic Security Profile 1.0.

7.2. WS-Security SOAP Header

WS-Security defines the Security SOAP header to carry security information in SOAP messages. Information included in this element includes, but not limited to, authentication credentials, digital signatures,

and encryption references. To specify security information for intermediary processing, use the actor element on the Security SOAP header. WS-Security specifies the wsu:Timestamp element as a child of the wsse:Security header.

Example 7.1. Sample of WS-Security

```
<soap:Header>
...
<wsse:Security>
  <wsu:Timestamp>
    <wsu:Created>2003-06-04T03:48:32Z</wsu:Created>
    <wsu:Expires>2003-06-04T03:53:32Z</wsu:Expires>
  </wsu:Timestamp>
  ...
</wsse:Security>
...
</soap:Header>
```

7.3. Authentication

The STAR Transport Group has selected Username/Password as the base method of authentication. The ability to authenticate via username and password is a base standard that all services must implement for the sake of interoperability.



STAR Level 1 Requirement

STAR1003 : All implementations are required to support Username/Password for authentication.

7.3.1. Username and Password

WS-Security defines a UsernameToken element to be used to pass the username and password. Below is the XML syntax of this element.

Example 7.2. WS-Security Username and Password

```
<wsse:UsernameToken wsu:Id="...">
  <wsse:Username>...</wsse:Username>
  <wsse:Password Type="...">...</wsse:Password>
  <wsse:Nonce>...</wsse:Nonce>
  <wsu:Created>...</wsu:Created>
</wsse:UsernameToken>
```

Two methods to include the password are supported:

1. Plain Text, in which the password is passed in clear text
2. Hashed, in which the password is not transmitted, but instead, a one-way hash is generated from the password and used for authentication

If a clear text password is used then it is required that the appropriate transport level encryption is used, such as HTTPS. All passwords must be stored or persisted in an encrypted format.

7.3.2. The Username element

The <wsse:Username> element carries the client identifier. For example, if the client is a dealer and the service provider is an OEM, the Username element will be the dealer's identifier. Different service providers require different types of identifications to identify their clients. Therefore, the syntax of this element is flexible and will be agreed upon between the two communication partners.

Example 7.3. Username Element

```
<wsse:Username>JohnDoe</wsse:Username>
```

Below are other possible examples on using the username field:

Example 7.4. Dealer Number

```
<wsse:Username>123456</wsse:Username>
```

Example 7.5. Unique ID that Identifies Dealer

```
<wsse:Username>JohnDoe</wsse:Username>
```

Example 7.6. Combination Dealer Number and ID

```
<wsse:Username>123456\JohnDoe</wsse:Username>
```

7.3.3. Plain Text Password

A password can be sent in clear text if a secure communication channel, such as HTTPS, is available between the sender and the receiver.

Example 7.7. Plain Text Password

```
<wsse:Security soap:mustUnderstand="1">
  <wsu:Timestamp>
    <wsu:Created>2003-06-04T03:48:32Z</wsu:Created>
    <wsu:Expires>2003-06-04T03:53:32Z</wsu:Expires>
  </wsu:Timestamp>
  <wsse:UsernameToken>
    <wsse:Username>JohnDoe</wsse:UserName>
    <wsse:Password
      Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">Password</wsse:Password>
    </wsse:UsernameToken>
  </wsse:Security>
```

7.3.4. Password Digest

When a secure channel is not available, or when the message goes through intermediaries, a password digest can be used to avoid revealing the password. WS-Security defines fields and algorithms to carry authentication information securely. The specifications use a one-way hashing algorithm, SHA1, to encrypt the combination of the password, a time stamp, the creation date/time, and a nonce (randomly generated string) to generate a digest. The resulting digest is base 64 encoded SHA1 hash value that is carried in the UsernameToken and verified on the server side.

Below is an example of a UsernameToken carrying a password digest:

Example 7.8. Password Digest

```
<wsse:Security soap:mustUnderstand="1" >
  <wsu:Timestamp>
    <wsu:Created>2003-06-04T03:48:32Z</wsu:Created>
    <wsu:Expires>2003-06-04T03:53:32Z</wsu:Expires>
  </wsu:Timestamp>
  <wsse:UsernameToken wsu:Id="SecurityToken-8a45f51b-fe46-4715-bdae-e596c36ad6be">
    <wsse:Username>JohnDoe</wsse:Username>
    <wsse:Password
      Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest">
      RvaxAmb2KhEQpFFJE+YXcyRy6E8==
    </wsse:Password>
    <wsse:Nonce>X6y15GC/WLYP8XY/YR3iIQ==</wsse:Nonce>
    <wsu:Created>2003-06-04T03:48:32Z</wsu:Created>
  </wsse:UsernameToken>
</wsse:Security>
```

The advantages of the digest method over the clear text method are:

1. Passwords are not transmitted over the wire
2. Since the Created element is included in the generation of the digest, the message recipient can reduce the risk of replay attacks by inspecting this element and rejecting messages that are older than a set time window.
3. To further reduce the risk of replay attacks the recipient can reject all messages that come with duplicate nonce values since nonces are generated to be unique. To accomplish this functionality, the server needs to store the nonce values of incoming messages for a period of time greater or equal to the expiration duration of the message, and compare the nonces of incoming messages to the stored ones.

There are situations in which a password digest cannot be used, such as when the password is not available to both: the client and the server (when using LDAP binding, for example).

7.4. Security Error Handling

The WS-Security specifications define a set of SOAP Fault codes to describe different error situations that may occur during the parsing of the security headers and authenticating or authorizing the requests. Sending a SOAP Fault back is not required because this could be used as part of a denial of service or cryptographic attack. However, if an error is sent back, it **MUST** use the SOAP Faults defined in the WS-Security specifications.

Here is a list of the fault codes as defined in WS-Security 1.0:

Fault Code	Description (Fault String)
wsse:UnsupportedSecurityToken	An unsupported token was provided
wsse:UnsupportedAlgorithm	An unsupported signature or encryption algorithm was used
wsse:InvalidSecurity	An error was discovered processing the <wsse:Security> header.

wsse:InvalidSecurityToken	An invalid security token was provided
wsse:FailedAuthentication	The security token could not be authenticated or authorized
wsse:FailedCheck	The signature or decryption was invalid
wsse:SecurityTokenUnavailable	Referenced security token could not be retrieved
wsu:MessageExpired	Security semantics are expired.

STAR Interoperability Rules

Level 1

STAR1015	STAR BOD Specific and Generic Transports must be message level interoperable.
STAR1001	All web services must be compliant to the rules and specifications outlined by the WS-I Basic Profile.
STAR1002	Appropriate compliance markers are required as specified by the WS-I Conformance Claim Attachment Mechanisms document.
STAR1016	Application level error messages MUST NOT be returned with a SOAP Fault, and MUST be returned using the appropriate BOD.
STAR1017	The service provider must keep track of contents that are deemed to have been received by the client to avoid resending.
STAR1020	The client must be able to handle duplicate messages from a service provider.
STAR1018	A SOAP Header MUST contain one manifest element for each content element in the SOAP body.
STAR1019	A manifest is REQUIRED to have namespaceURI , element , contentID , and version attributes. Even though version is listed as optional it is REQUIRED for STAR BOD and DTS transports.
STAR1009	All STAR Web Services are REQUIRED to understand and handle the STAR Specific SOAP Faults.
STAR1010	All STAR soap fault error codes are REQUIRED to be prefixed with STAR: and the appropriate STAR error code. i.e. STAR:Invalid Structure
STAR1011	All STAR soap fault error codes are REQUIRED to appear in the standard SOAP:Fault block.
STAR1012	SOAP Faults are for Critical Processing errors only. Informational or warning errors should not be sent as a SOAP Fault.
STAR1014	WS-Security errors must send the appropriate WS-Security SOAP Fault for the authorization being used.
STAR1013	ConfirmBOD reason codes that are sent at the Warning or Informational status, SHOULD NOT trigger a resending of the BOD.
STAR1004	All implementations are REQUIRED to send information over HTTPS.

STAR1008	All services and clients must be compliant to the general Security requirements outlined by the WS-I Basic Security Profile 1.0.
STAR1003	All implementations are required to support Username/Password for authentication.